Spring 2020

Auto-Landing System for Fixed-Wing Unmanned Aerial Vehicle

Dan Dermer
San Jose State University

AUTO-LANDING SYSTEM FOR FIXED-WING UNMANNED AERIAL VEHICLE

A Thesis

Presented to

The Faculty of the Department of Computer Engineering

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Dan Dermer

May 2020

The Designated Thesis Committee Approves the Thesis Titled

AUTO-LANDING SYSTEM FOR FIXED-WING UNMANNED AERIAL VEHICLE

by

Dan Dermer

APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING

SAN JOSÉ STATE UNIVERSITY

May 2020

Dr. Donald Hung, Ph.D.	Department of Computer Engineering
Dr. Wencen Wu, Ph.D.	Department of Computer Engineering
Mr. David Sadighi, B.S.	Senior Staff Electronics Engineer, Lockheed Martin Space
Dr. Joseph R. Buck, Ph.D.	Senior Fellow, Lockheed Martin Space

ABSTRACT

AUTO-LANDING SYSTEM FOR FIXED-WING UNMANNED AERIAL VEHICLE

by Dan Dermer

Unmanned aerial vehicle systems, or any kind or autonomous system, are relevant to many applications today. However, they are complex and sophisticated systems that require a deep understanding of multiple technologies. In addition, the mathematical rigor, computer modelling, and programming applications involved make this a challenging field of study. This thesis explores the possibility of achieving the automated landing of a fixed-wing unmanned aerial vehicle. Auto-landing systems can resolve the challenges for the novice user and make aerial vehicle platforms accessible and dependable. A wide spectrum of applications such as agriculture, aerial photography, and security, to name a few, can utilize this technology. This thesis catalogs, describes, and analyzes the research into existing solutions, attainable technologies, and the process used to develop and validate a control algorithm that can land an airplane safely.

ACKNOWLEDGMENTS

This SJSU Master's Thesis is the culmination of the efforts of several individuals. I would like to express my gratitude to my academic advisor Dr. Donald Hung for his support, guidance and patience throughout my research. I want to expresses my deep appreciation to my technical advisor David Sadighi for his insightful guidance, substantial knowledge, and for spending countless hours enthusiastically discussing control theory with me.

I would especially like to thank Stephanie Bures for her support in my journey towards this degree, her advice, enduring patience, guidance, and for her many hours spent editing and proofreading. I would like to thank James Mack for his helpful attitude no matter the challenge, and in particular his assistance with the sophisticated auto-pilot and simulation environment. I would like to thank Joel Wiser for his contribution in the early stages of the hardware integration, for suppling some of the more expensive parts of this system, and for participating in the freezing weather test flights. I would also like to thank the members of my thesis committee: Dr. Joseph R. Buck for his valuable time and for his thoughtful review of my thesis draft; Dr. Wencen Wu for her constructive feedback during the final stages of this thesis. Last but not least, I want to thank my parents and family for their love and support given me throughout my life.

Their contributions made this effort to be what I hope is a respected reference for this new technology. Without the support of each of these individuals, this research wouldn't have been such a positive and rewarding experience.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

A control algorithm for the automated landing of fixed-wing airplanes presents a challenging problem requiring multiple approaches. One of the approaches is a control feedback system utilizing a sophisticated embedded system, a stabilization system, and sensors. All three technologies of interest are required in order to design a platform that is capable of flying autonomously to a desired location and performing a safe controlled landing. These capabilities are still cutting-edge technologies and, as such, require further development. There is debate in the industry as to the utility and range of potential applications for unmanned aerial vehicle technologies.

In addition to the relevance of current unmanned aerial vehicle (UAV) technology development, the field of flying robots has shown dramatic growth in the last few years. Consequently, there is a massive amount of information on the subject. Unmanned aerial vehicles have multiple target uses, including aerial photography, surveillance and monitoring, merchandise delivery, and weather monitoring. These capabilities were not widely available in the commercial industry just a few years ago. The development of new types of batteries, power systems, communications systems, and small but powerful microcontrollers enables new applications for UAVs.

Unmanned vehicle system development requires proficiency in electronics and programming, plus a background in embedded systems, sensors, control systems, linear algebra, and general mathematics. To define an appropriate

scope, a combination of an off-the-shelf auto-pilot controller and a dedicated auto-landing controller were selected. This approach divided the tasks of flying and landing between the two controllers. The benefit of this approach was, first, to simplify the thesis development in order to focus on the landing portion only, and secondly, to optimize each controller capability for their dedicated task. The auto-landing controller was an ARM M3 based CPU, and it was equipped with appropriate interfaces such as pulse width modulation (PWM) to interact with servo motors typically utilized on unmanned aerial vehicle platforms. Additionally, this controller can interact with the auto-pilot over UART. The auto-pilot controller, on the other hand, was capable of flying the aircraft and following pre-programed way-points, but it lacks the capability to run dedicated C++ firmware hosting a control loop algorithm in order to control actuators or servos.

Because a significant challenge to UAV pilots is the safe landing of the vehicle, the employment of an automated system could provide a repeatable and reliable means to safely land an expensive commercial or military asset. After active engagement by the UAV pilot, this automated landing system would be capable of managing variable lighting conditions such as no light or low light thereby, minimizing the probability of a catastrophic crash due to human error. This capability would potentially open up the mission space of UAV technology.

An automated landing system can be a challenge to develop given the requirement to balance and manage multiple hardware, software, and environmental variables. The system must have the capability to “read and react”

in real time. Multiple sensors must be read simultaneously, such as for altitude, attitude, heading, position, and speed. Once interpreted, the system must “react” by operating an airplane independently without the need for pilot inputs. Control loops, sensors, and robust embedded microcontrollers can be employed to manage vehicle telemetry and affect the wing geometry changes required to safely land the aircraft.

The expectations from the system that were developed as part of this thesis research were limited to a system that is be able to operate in equable weather conditions. Wind gusts of a few miles per hour can be tolerated, but stronger winds are beyond the current capability of this system. The airplane should be able to land with an altitude accuracy of ±1 foot, and then transition to the flare mode using Lidar for higher accuracy.

Many sources for control systems for unmanned vehicles were encountered during the literature review process, with most of the sources concentrating on quad-copter control and navigation. Few sources were available that focused on automated landing systems for fixed-wing airplanes. Control systems, sensors, and auto-pilot systems were a focus of the initial research required to understand the current state of the art and limited the present scope. Papers on control systems covered attitude estimations and stabilization, elevator altitude, and detailed design of fixed-wing configurations. Papers focused on sensor technology described optical flow sensors, data fusion, and partial sensor data. Visual navigation, autonomous landing, and autonomous flight in general

were the focus of the research on auto-landing flight systems. Compatible articles provided a useful means for shaping the design, framing the scope, and illuminating the expected challenges.

The control algorithms are computationally intensive and require complex modeling to understand and manipulate the many variables the vehicle would be seeing during operation. MATLAB modeling was used for the development of the mathematical algorithm with C++ for the implementation of the algorithm into the microcontroller. The microcontroller calculates data from multiple inputs and provides outputs in real time, thereby controlling elevators to affect the safe landing of the fixed-wing UAV.

1.1 Thesis Outline

This thesis is structured in the following manner: introduction, problem definition, hardware and software integration, control systems research, controls development, and conclusions. Fig. 1 details the individual steps.

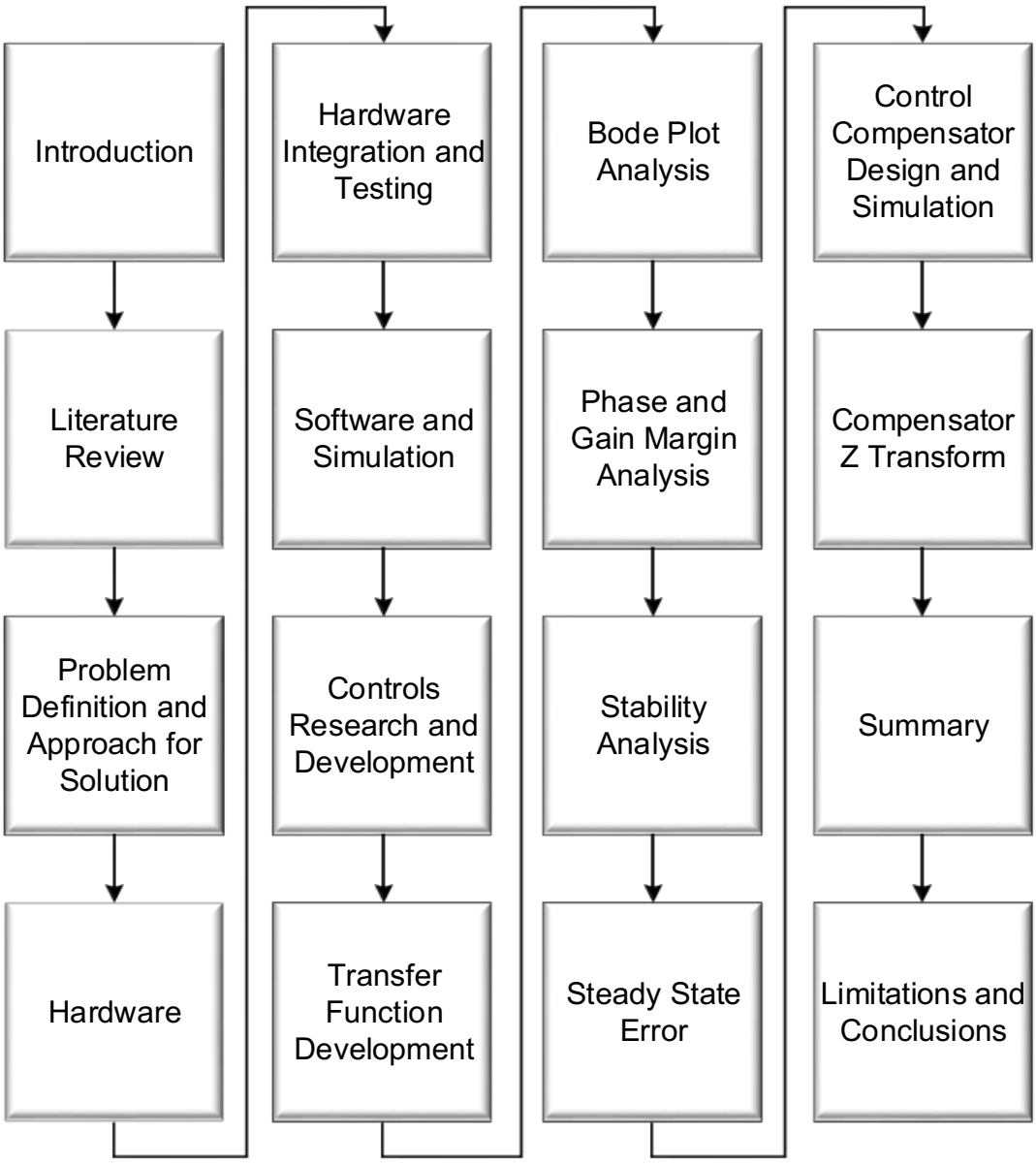


Fig. 1. Thesis outline.

2 LITERATURE REVIEW

The popularity of UAVs has increased dramatically in recent years. A reduction in price of unmanned vehicle platforms, as is typical as technology incrementally improves, has made it possible to design and implement different types of applications relatively affordably. However, the technical challenges and complexity of the design configurations still allow rapid developments, improvements, and specialization. This review of current literature covers several articles that detail different approaches for the development of a UAV system. Existing solutions must be researched in order to develop a new approach.

2.1 Control Systems

The unmanned vehicles control system is arguably one of the biggest challenges in maturing and specializing this technology for various applications. Several articles have analyzed various control system technologies and methodologies. Controlling unmanned vehicles can be done in many ways, as detailed by Zhang and Wang [1] who developed and demonstrated a hierarchical control system for fixed-wing UAV landing. The paper describes a system that was developed with a layered system in mind, including top level, mid level, and low level control systems. The landing was also divided into three stages: descent, flare, and taxiing. Additionally, each level was designed to handle different kinds of disturbance rejection. The advantage of such a system is the optimization of each layer to the corresponding landing stage, rather than one system that will attempt to handle all of the landing stages.

Although Zhang and Wang's scope went beyond a one-dimensional autonomous landing, the paper is still relevant, as the main goal coincided with a fixed-wing auto-landing. Dividing the landing into stages is a useful concept. This approach makes the control system less complex, as it removes the requirement to manage the flare portion of the landing.

Amit and Padhi [2] researched autonomous landing using tracking-model predictive static programming guidance and dynamic inversion auto-pilot. Their approach was similar to this thesis as it followed a desired trajectory landing path. There were, however, a few differences between the two approaches. While the approach of this thesis was to use experimental data to compute the transfer function of the plane, Amit and Padhi used a state space theoretical model with six degrees of freedom. Another difference was the number of control loops employed in the respective models. Amit and Padhi used an outer and inner control loop, with the outer loop optimized for the control of the deviation across the landing trajectory and the inner loop tuned for the implementation of the dynamic inversion technique, control of body rates, and forward velocity. Lastly the treatment of the landing phases differed as they sectioned the landing into phase circular orbit, glideslope, and flare. This paper was largely theoretical but mathematically comprehensive. The goals of both papers were similar, though the approach was rather different.

Cheng and Zheng's [3] utilized an elevator-only control for landing a UAV. Their approach was similar to this thesis methodology, as both methods

controlled only the elevator to follow a desired trajectory. Cheng and Zheng's objective was to insert the plane into a deep stall state where the lift drops and the drag increases. This is a unique method of landing and is not commonly used. A more straightforward approach of following a pre-loaded trajectory was used in this thesis. Following a trajectory that was developed from experimental landing data guaranteed that the airplane would not reach a state of stall at any point. Cheng and Zheng's concept was an interesting one, and it may be beneficial to investigate it further as an improvement or alternative.

Brezoescu, Espinoza, Castillo, and Lozano's [4] described a focused treatment of plane lateral dynamics. Both constant and variable wind disturbances were discussed and a closed loop adaptive control system was developed to maintain a desired trajectory in the presence of crosswind. They defined an approach for simplification of a fixed-wing model, related degrees of freedom and constraints, and supported the use of an external auto-pilot to reduce the control system complexity. The adaptive control loop employed by these authors could potentially add valuable robustness to a simplified solution and is a candidate functionality for future incorporation.

Qiang and Junqing [5] examined control system design and validated small and low cost systems. They discussed a candidate application for a low cost and low weight system. Therefore, their approach was useful as it was a similar to a small system control design.

Jiang, et al. [6] elaborated on the system structure and details of the critical subsystems. They also analyzed dynamics of flying the robotic UAV in near hovering mode. This aligned well with the subsystem structure approach in this thesis work.

Senkul and Altug [7] presented a different control system. Their alternative system used tilting rotors instead of frame tilting. Although potentially better performance could be achieved with this design for quad-rotors, this article was less informative as it is not applicable to an airplane.

Choi et al. [8] discussed formation, or swarm, control of a system of up to four quad-rotors. A formation control law was derived and covered in detail. While control of a system of multiple quad-rotors was not the main focus of this thesis, this paper also contained a general discussion of control problems in three dimensions. Because a foundational understanding of multidimensional control to generate simulations and mathematical models is critical for aerial vehicle control systems, this reference may be useful for a single vehicle system as well. Stable simulation results for formations of four quad-rotors in three dimensions were also presented, which can be useful as an additional source of reference and practical data.

Jeong et al. [9] presented a small quad-rotor system designed for educational purposes. This approach may be easier to understand for novice designers and engineers who do not have a strong background in such applications. The foundational aspects of this research make it a good educational resource. A few

other advantages are that the authors were conscious of budget restrictions faced by many novices and educational institutions in addition to pursuing limited problems in a limited manner. Furthermore, the control unit in the quad-rotor system was an AVR 8-bit controller. The use of a simple CPU in their system mandates shorter algorithms for faster calculations. Shorter algorithms are easier to understand and implement, making them ideal for a first attempt and UAV design.

2.2 Sensors

While control systems form the “brains” of autonomous aircraft systems≠ the sensors in an aircraft system can be considered as the “eyes,” where the telemetry inputs are collected to facilitated the controls. Without the sensors (*e.g*., position and light detection), aircraft systems would have very limited applications and response capabilities. A few articles that discussed different types of sensors, such as GPS based, optical flow, radar and others that use partial sensor data analysis were reviewed. The variety of approaches detailed using different sensing theories was good source of information for an unmanned aerial vehicle control system project. Analyzing multiple technologies increased the probability of finding a technology that can be utilized by this project requirement and budget successfully.

Yoo et al. [10] developed a novel, affordable GPS component solution for the small UAV. To overcome the constraints typical of simple GPS based sensors, such as signal interrupts from obstacles or signal jamming, they present a

solution using a GPS/INS (Inertial Navigation System) sensor fusion. This technology has particular relevance for this common and debilitating signal dependency and can potentially improve the performance of autonomous design while remaining within the budget constraints of a hobbyist or commercial user.

Lim et al. [11] considers using strap down optical flow sensors. This technology is useful when using an airframe with limited payload carrying capabilities, because they are light weight. These sensors are of particular interest depending on the chosen platform configuration.

Chan et al. [12] analyzes the integration of a 16-bit embedded microcontroller with a data fusion sensor system. This system can be used as an optional anti-drift mechanism in auto-pilot design. The paper discussed an algorithm and sensor configuration that can resolve a drift caused by vibrations and noise measurements and both cause control performance problems. The authors indicated good results in their testing and detailed how the system used a data fusion sensor to overcome disturbances.

This research paper by Kis et al. [13] discusses emergency situations that would typically be encountered by a UAV system that would result in having to utilize fewer sensors than in normal operation. The sensors could be lost due to malfunction or damage. Solutions involved the implementation of two types of control algorithms: “causal LQ state feedback” and “H-control.” These compensation techniques may not be necessary in a typical design as they are more advanced and allow for component redundancies. However, the features of

the presented design could be of benefit in some projects, as they involve configurations using fewer sensors than ideal for cost reasons or for the benefit of power conservation and efficiency.

Kwag and Kang's work [14] can be interpreted as a review of several different sensors. In particular, the radar sensor fits the size constraints that a small design requires. The low altitude flight characteristics of this radar sensor is another aspect that matches small scale project design requirements well. This type of sensor provides a few more advantages such as all-weather environment acquisition capability.

2.3 Auto-pilot System Landing

Auto-pilot algorithms combine the control and sensor capabilities and use them to autonomously control the aircraft. There are many ways to develop an algorithm for auto-piloting, and there are different auto-pilot implementations. This literature review focused on the landing portion of the auto-pilot algorithm. The following articles covered different approaches to achieve the same goal.

Swami et al. [15] narrowed their subject down into the development of a soft-landing quad-rotor algorithm. The algorithm to control the soft landing of the quad-rotor system in real time via changing its orientation may be useful even when utilizing other kinds of sensors. The implementation of this algorithm may prove to be an important addition to an autonomous landing project, as the continuous impacts of rough landings may damage the expensive equipment.

The research paper by Campbell et al. [16] discussed the problem of coordinating the movements of several unmanned aerial vehicles, as such it can be more relevant for multi vehicle projects. The OptiTrack optical array solution offered in this article, however, is an interesting technology and would be potentially effective in control methods or guides.

Kendoul et al. [17] researched a method for controlling unmanned aerial vehicles using visual navigation. They were able to present and validate results from autonomous systems using this method. This could be adopted in a design as an alternative solution for auto landing in a target area where GPS may not be sufficiently accurate. Auto-landing a vehicle in a precise geographical area and variable surface is a real challenge.

Xiang et al. [18] presented an economical control solution that is capable of autonomous takeoff and landing. They used a Wii remote camera combined with an infrared camera. This is an interesting approach that could be utilized in a design. It is both simple and inexpensive. For these reasons it could be a valuable configuration to consider reducing cost and complexity.

Herisse et al. [19] used an extremely simple unmanned aerial vehicle system equipped with just a CPU and camera. They managed to perform stable hover and automatic landing maneuvers using divergent optical flow as a feedback loop. The algorithm used in their paper may be useful for control loop development. The usage of the inertial optical flow sensor as described in this paper has been used by other researches with similarly reliable results.

3 PROBLEM DEFINITION AND APPROACH FOR SOLUTION

3.1 Defining the Problem and Approach for a Solution

Auto-landing is a challenging task with many possible variables. Attempting to solve all aspects of autonomous flight and auto-landing was beyond the scope of this thesis. The approach taken was to limit the problem to the design of a control loop that would execute a safe descent by following a preprogrammed trajectory. This method is valid provided that the GPS coordinates and attitude of the runway are known, as shown in Fig. 2. The algorithm will always begin following its preprogrammed trajectory at the same speed, distance and altitude from the runway start.

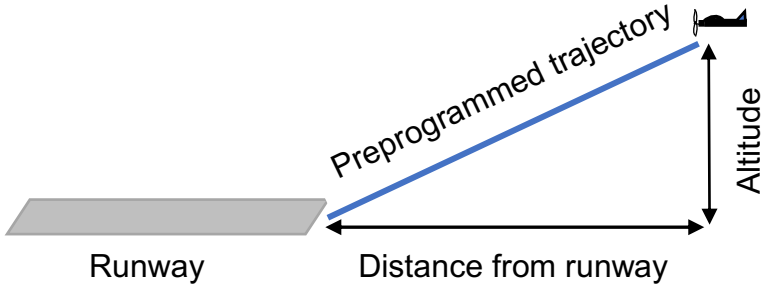


Fig. 2. Preprogrammed landing trajectory.

This system can work for a variety of applications such as military drones, research UAVs, hobby airplanes, and for a landing assistance system for full scale airplanes.

With this approach in mind, a search was started for available hardware components appropriate for an application with size, weight, and budget constraints. The auto-pilot chosen for this project enabled stable autonomous

flight with navigation capabilities. The auto-pilot utilized an array of sensors that are logged during flight. The logged data were a key to the development of this system as it was used later to develop a transfer function. Additionally, a microcontroller was used that could host the control algorithm and read the auto-pilot telemetry. Other components selected were two multiplexers that enabled bypassing the auto-pilot such that the aircraft could be variably controlled by either the microcontroller, the auto-pilot, or manually by a pilot.

After successful integration of the auto-pilot for autonomous flight of the airplane, the focus of the effort transitioned to development of the transfer function of this air frame. A trade was investigated to select between using a generic airplane transfer function versus deriving the actual transfer function of this specific airframe. While a generic transfer function would have been easier and quicker to utilize, the limitations involved possible inaccuracies in the model and less than optimal performance of the control loop. Therefore, a unique airframe specific transfer function was developed.

Experimental flight tests were conducted to inform the creation of the transfer function using the auto-pilot as the data logging device. The data of interest were the input elevator and output altitude. The relationship between the two provided the information required to build a transfer function. Two different methods were employed in this process. The first was to try to fit a function manually by inspecting the input and the output. The attempt to fit an integrator did not produce satisfactory results.

The second method was to use the MATLAB “System Identification Toolbox.” This tool can fit a transfer function for a given input and output. After several iterations with the MATLAB tool a transfer function with better than 95% fit was generated. The next steps were to analyze the function and design a compensator for satisfactory open loop stability (further discussed in the Section “Phase and Gain Margin Analysis”) and closed loop tracking and step response. Simulating the closed loop function produced the desired landing trajectory.

3.2 Research in Other Available Hardware Capabilities

An important step in the development of any system is to research the state and availability of current technologies. It is important to know the capabilities and limitations of those technologies before developing an algorithm. Research was conducted as part of the literature survey to collect a list of options.

3.3 Defining the Concept of Operation and the Algorithm

Auto-landings generally follow the scenario shown in Fig. 3. When the auto-landing microcontroller identifies the desired beginning of the landing location using MavLink serial bus telemetry reading from the Pixhawk auto-pilot, the landing algorithm is initiated. The auto-landing controller takes over the elevator servo and the ESC motor speed control via the servo multiplexer.

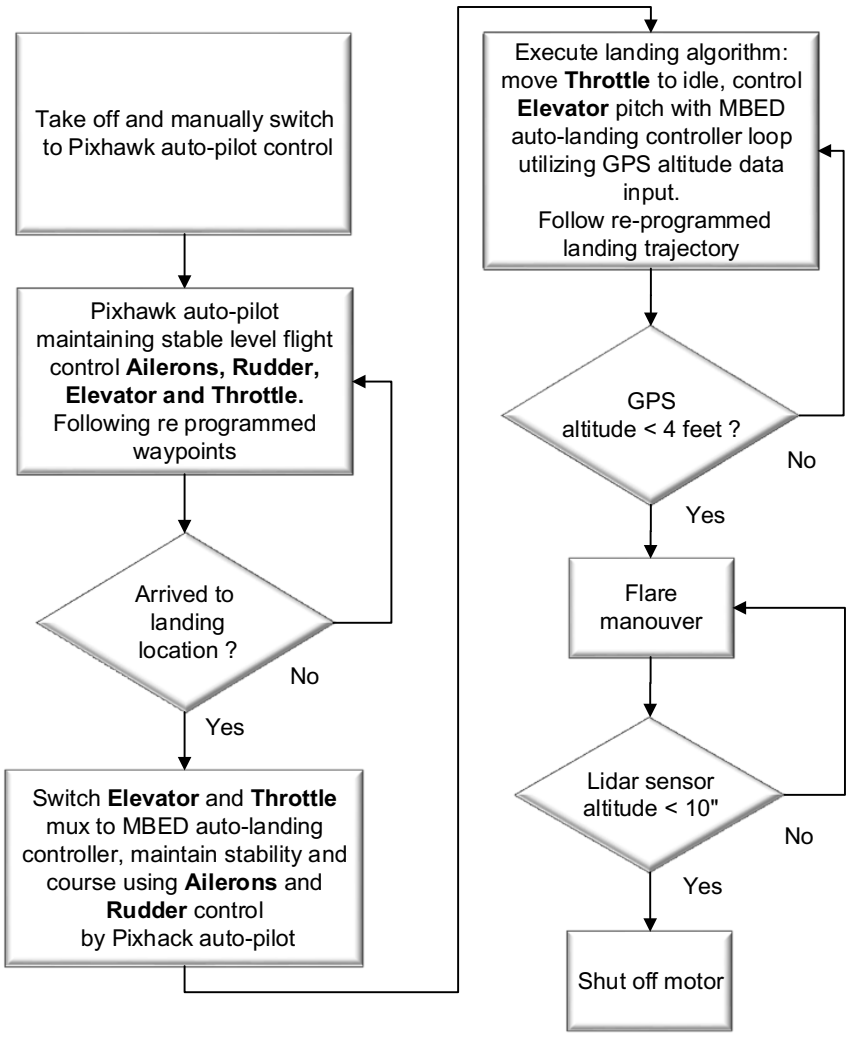


Fig. 3. Diagram detailing auto-landing firmware flow.

Control over the rudder and aileron servos remains under Pixhawk auto-pilot to maintain stability and heading while descending. The approach taken in this thesis was to decouple the landing-control loop from the Pixhawk auto-pilot algorithm. When the landing algorithm was initiated, the assumption was that the stability of yaw and roll was maintained, as the Pixhawk auto-pilot continuously controls the aileron and rudder.

The first and most critical data inputs used for descent algorithm development are altitude and air speed. Altitude is read via the GPS sensor, Lidar proximity sensor and air speed via the air speed sensor. The control unit adjusts the motor and the aircraft elevator to elicit the desired angle of attack as the speed adjusts to the required glideslope, or trajectory. As a starting point, a control loop was investigated for use in controlling the aircraft descent (Fig. 4).

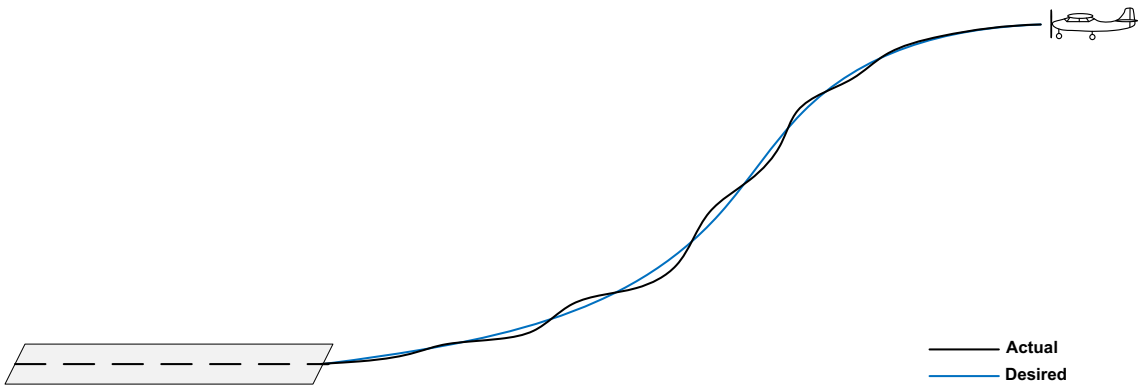


Fig. 4. Aircraft glideslope.

Fig. 5 shows the approach taken to integrate the complete system. The left side of the diagram describes the auto-landing MBED ARM M3 controller hardware and its firmware software. The right side describes the airplane auto-pilot Pixhawk ARM M4 controller, sensors, multiplexers, and radio control. The control loop algorithm is embedded in the auto-landing controller software and with inputs from the hardware sensors.

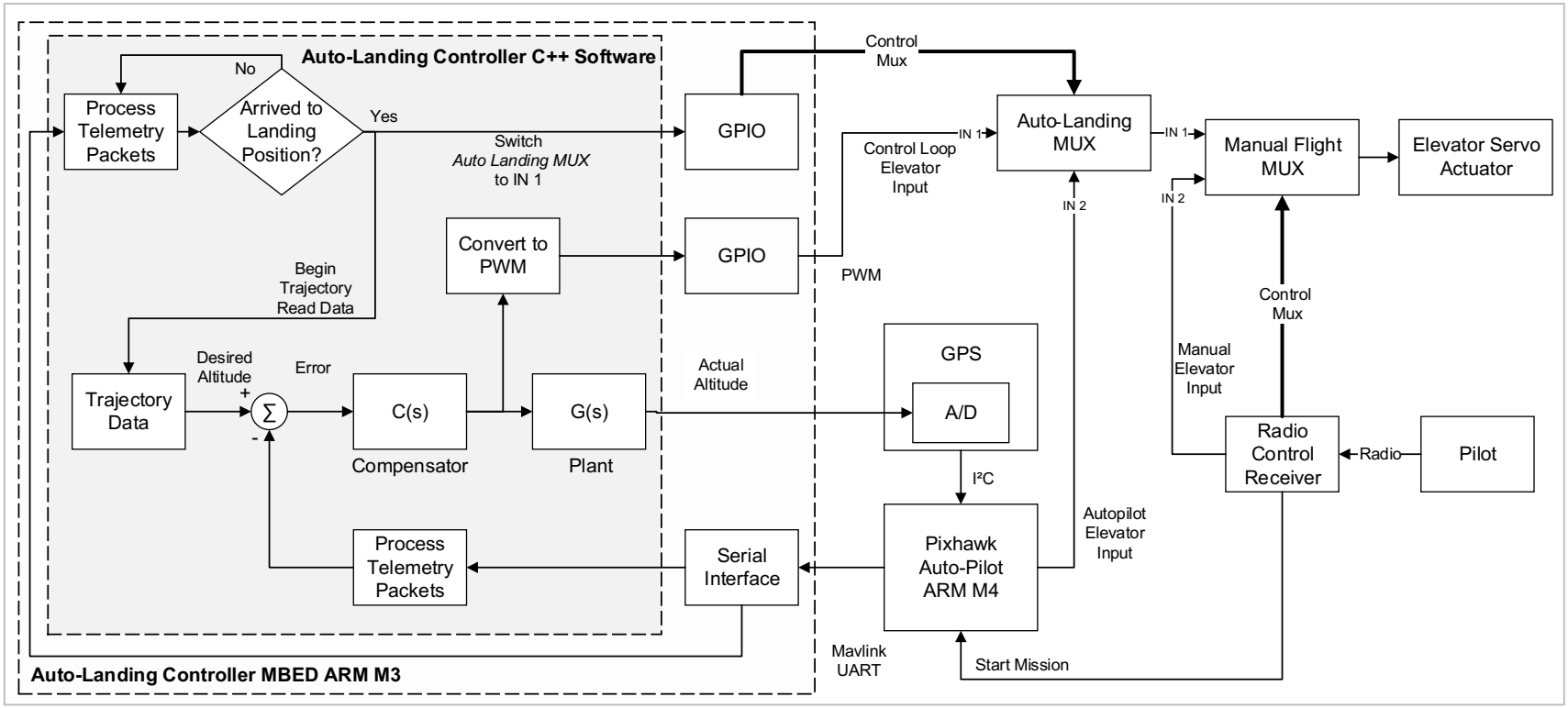


Fig. 5. Software hardware and algorithm integration.

4 HARDWARE

This chapter outlines the hardware used in the development of the fixed-wing auto-landing system. It describes each component's functionality, integration, and interfaces. Appendix A contains a detailed specification table for each component.

4.1 Airframe

The airframe that was chosen for this project, shown in Fig. 6, is the E-flite Apprentice S 15e RTF. This airframe offers durable structure, good flight

Fig. 6. Air frame – E-Flite Apprentice S.

characteristics, and an adequate amount of internal equipment space that can be used for the control equipment.

4.2 Auto-Landing Controller

The auto-landing controller is responsible for hosting the landing algorithm software. This controller is configured to automatically take control of the elevator from the Pixhawk auto-pilot via an external multiplexer, as detailed in Section

“Servo Multiplexer.” The San Jose State University Computer Engineering program is most experienced with the NXP-LPC controller, which was a primary factor in choosing this platform. The MBED platform is a development board with an NXP-LPC1768 microcontroller designed for prototyping. Fig. 7 details the MBED platform including all its peripheral interfaces.

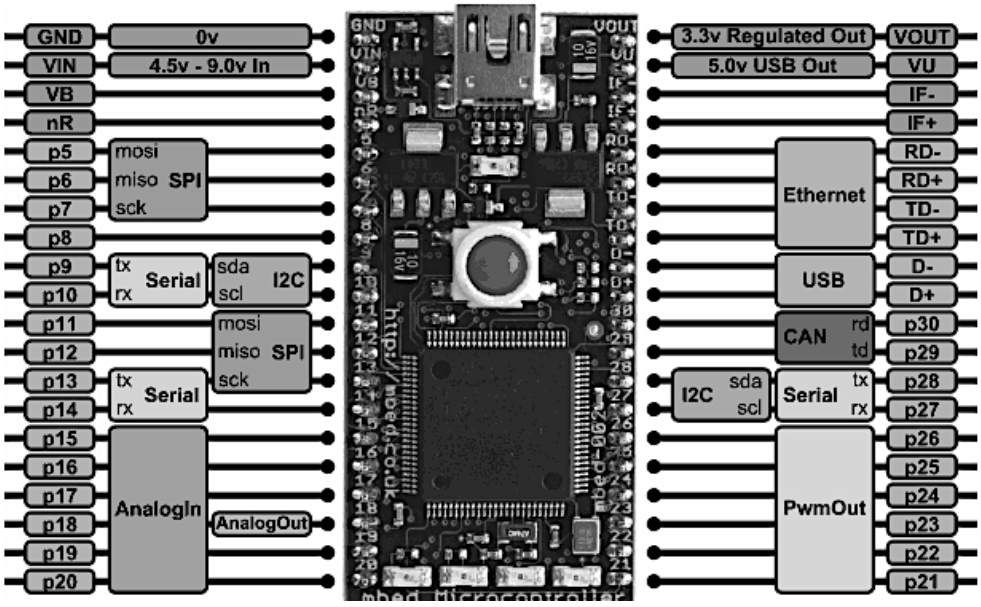


Fig. 7. Auto-landing controller, MBED NXP LPC1768.

The NXP LPC1768 is a 32-bit ARM Cortex-M3 core running at 96 MHz. It includes 512 kB flash, 32 kB RAM and interfaces such as ethernet, CAN, SPI, I2C, ADC, DAC, PWMs, and USB host. This project used the A/Ds, the serial UART, and the PWMs interfaces.

4.3 Auto-Pilot Controller

The Pixhawk auto-pilot in Fig. 8 is the component that functions as the flight control of the airplane. It enables navigation and affords stability of the airplane.

Fig. 8. Auto-pilot, 3DR Pixhawk.

In this project, the auto-pilot is utilized to fly the plane towards the landing position. The auto-pilot transmits flight location, speed, altitude and heading telemetry. When the “controlled descent” starts the auto-pilot maintains flight

stability and heading. This 3DR Pixhawk is an open source device and is very common and well known in the drone enthusiast communities.

4.4 Sensors

4.4.1 GPS

The GPS module shown in Fig. 9 was used by the Pixhawk auto-pilot to determine the location of the aircraft in space. The requirements for this sensor were low power consumption (<100 mA), low weight (<30 g), and a small foot print (<40 x 40 x 10 mm).

Fig. 9. 3DR GPS with compass.

The 3DR GPS sensor was chosen as it met all project requirements, and it was recommended by the auto-pilot manufacturer. Although this specific GPS sensor was preliminary designed to work in an integration with the Pixhawk auto-pilot system, the control system in this thesis utilized the sensor as a stand-alone altitude input. The GPS sensor altitude data were read through the auto-pilot Pixhawk MavLink telemetry bus. After several experiments, it was clear that the data rate of this sensor would be a bottleneck for this control loop. The

sensor update rate is only 5 Hz. This limited the bandwidth of the control loop to only few hertz. This low bandwidth was insufficient to track a typical landing trajectory and caused a steady state error. Tuning the control loop compensator made it possible to overcome this shortcoming and resulted in a small steady state error. A faster sensor would make this control system track the input set-point more accurately.

4.4.2 Proximity Sensor

The proximity sensor shown in Fig. 10, was used to measure the distance between the airplane and the ground while in flare mode. The sensor that was chosen for this project was made by PulsedLight (now Garmin International). The requirements for this sensor were low power consumption (<200 mA), low weight (<30 g), a small foot print (< 40 x 50 x 25 mm), range of at least 1 m, and accuracy of better than 5 cm.

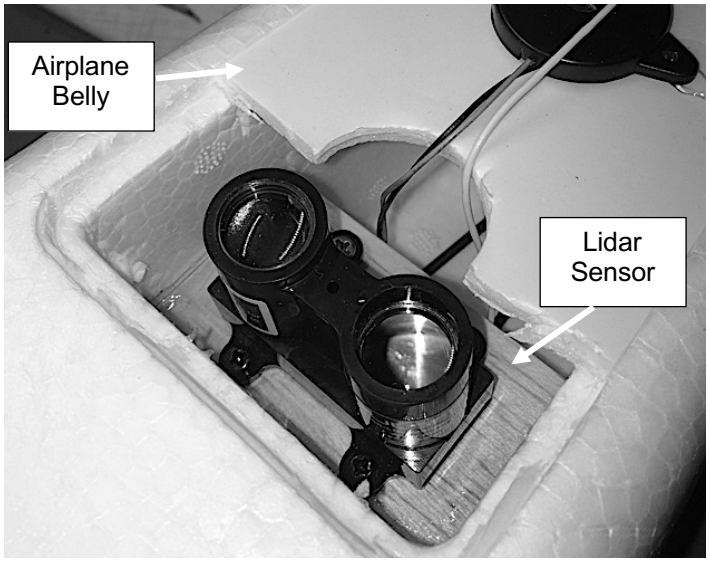


Fig. 10. Proximity sensor, Lidar Lite.

Fig. 11 shows the advantage of the Lidar sensor over a GPS sensor for small ranges, as it has high resolution and can measure altitude in centimeters. High resolution measurement can help during the flare stage and land the airplane gently as the distance from the runway decreases to a few inches.

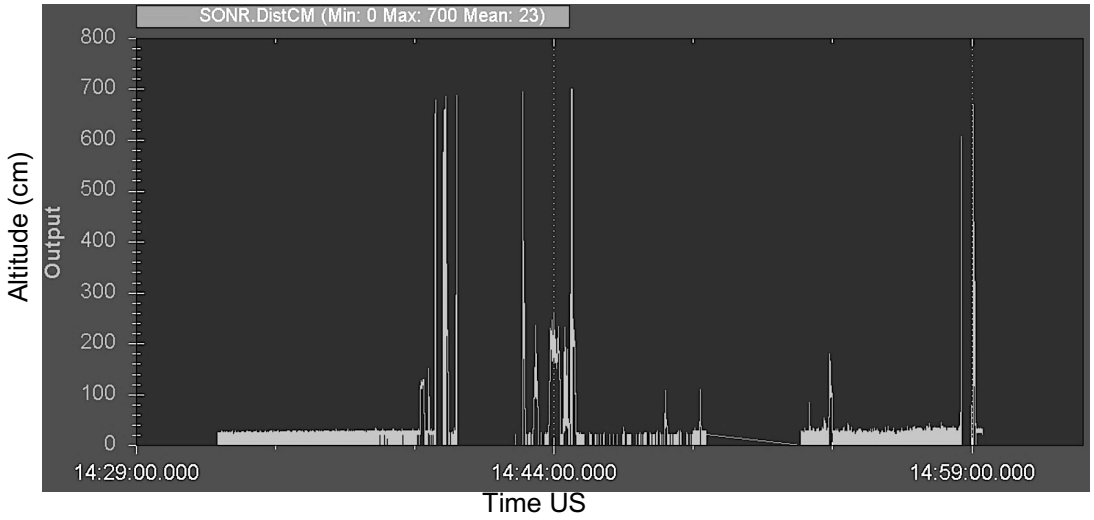


Fig. 11. Lidar proximity sensor readout during a flight.

Although the Lidar sensor seemed to work in a satisfactory manner on all tested terrains, Fig. 12 shows the limitation of the Lidar sensor as it was incapable of measuring distances over several feet.

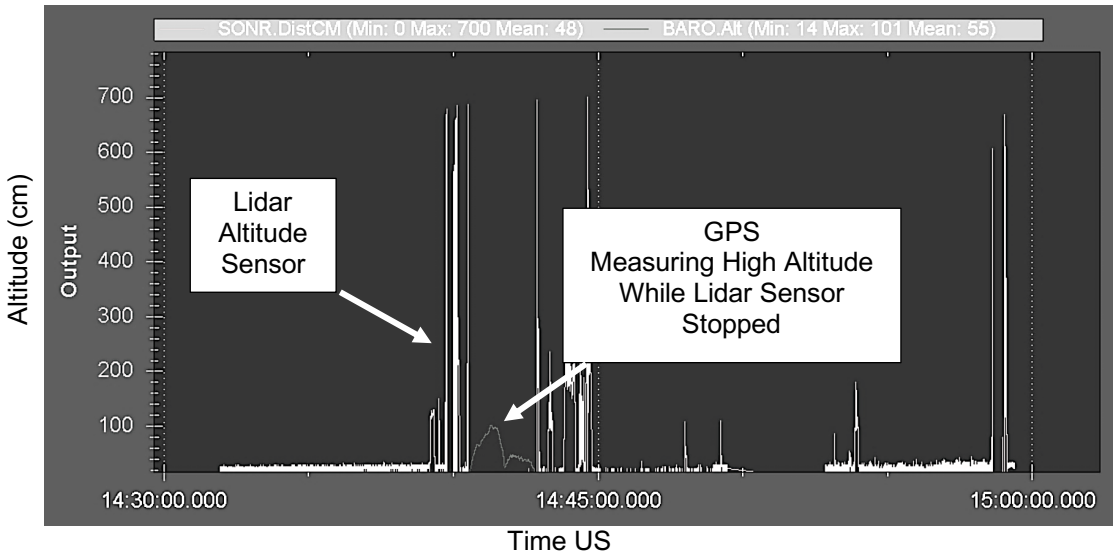


Fig. 12. Lidar *vs.* GPS altitude measurements.

4.4.3 Air Speed Sensor

The air speed sensor shown in Fig. 13 was used to measure aircraft speed. The speed was required by the Pixhawk auto-pilot for its internal algorithm. This sensor was made by 3DR and utilizes an I2C serial connector. The sensor also measures temperature to allow for the correct bias of true airspeed from indicated airspeed using the MS5611 static pressure sensor on Pixhawk. GPS airspeed was used in lieu of this sensor for the landing-control loop algorithm.

Fig. 13. Air speed sensor, 3DR digital air speed sensor.

4.5 PPM Encoder

Fig. 14 is the Pulse Position Modulation (PPM) encoder. It allowed the connection of multiple PWM servos via a single connector. This feature reduced the number of wires between the Radio Control Receiver and the Pixhawk auto-pilot.

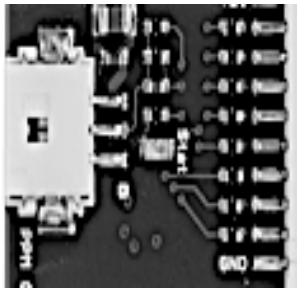


Fig. 14. 3DR PPM encoder.

4.6 Servo Multiplexer

The 4-channel multiplexer from Pololu shown in Fig. 15 allowed the aircraft controls to be switched between two sources. In this application two multiplexers

were used. One was used to switch between the Pixkhawk auto-pilot and the MBED auto-landing controller when a landing procedure started. The second multiplexer was used to switch between autonomous and manual flight modes. This functionality allowed a pilot to override the auto-pilot by switching to manual via a remote-control radio transmitter.

Fig. 15. Servo mux, Pololu 4 channel radio control multiplexer.

4.7 Motor Speed Controller

Electric Speed Controls (ESC) were used to convert a PWM input signal, such as output from Radio Control Receiver, into a high voltage and high current motor input. The E-flite 30-Amp Pro SB Brushless ESC Speed Controller shown in Fig. 16, was used in this project. This Speed Controller was recommended for this motor and the airframe by the manufacturer.

Fig. 16. Motor speed controller.

The E-flite 30-Amp Pro features up to 30-amps of continuous current with proper airflow, 35-amps peak with a 5-volt Switch-Mode BEC circuit capable of 700 mAh continuous current. This ESC was capable of driving up to 5 analog or 4 digital sub-micro servos.

4.8 Motor

The E-flite BL15 EFLM7215 Brushless outrunner motor (840Kv) shown in Fig. 17, was recommended by the airframe manufacture.

Fig. 17. Brushless outrunner motor.

4.9 Battery

The E-flite High-Power Lithium Polymer Pack battery shown in Fig. 18 was recommended by the airframe manufacture.

Fig. 18. Lithium polymer 3200 mA battery.

Fig. 19 demonstrates a flight of approximately 8 minutes. During this flight, the airplane systems consumed 970 mA of total current. This consumption aligns well with the battery requirement capacity of 3200 mA. The capacity of 3200 mA should be sufficient to provide an average flight time of 20 to 25 minutes. Although Fig. 20 shows that under certain loads the battery voltage dropped to less than 11 v, this system was designed to work with minimum of 6 v input.

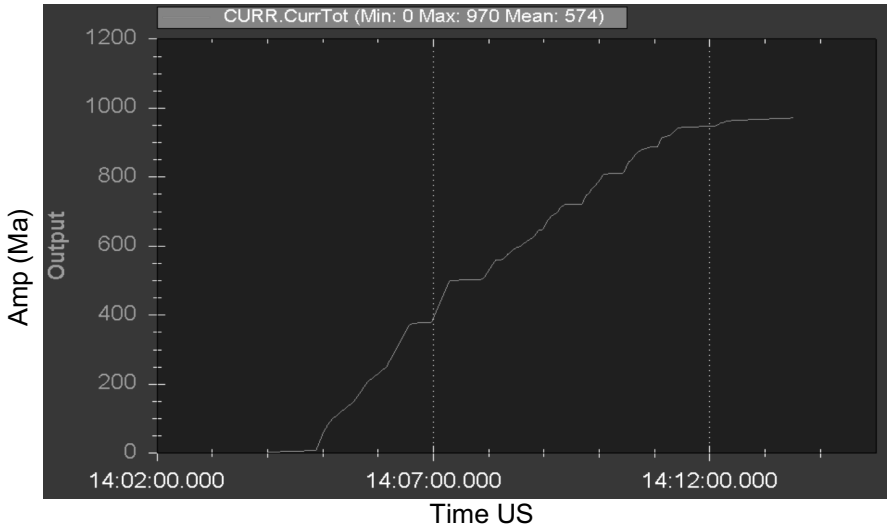


Fig. 19. Current consumption over a complete test flight.

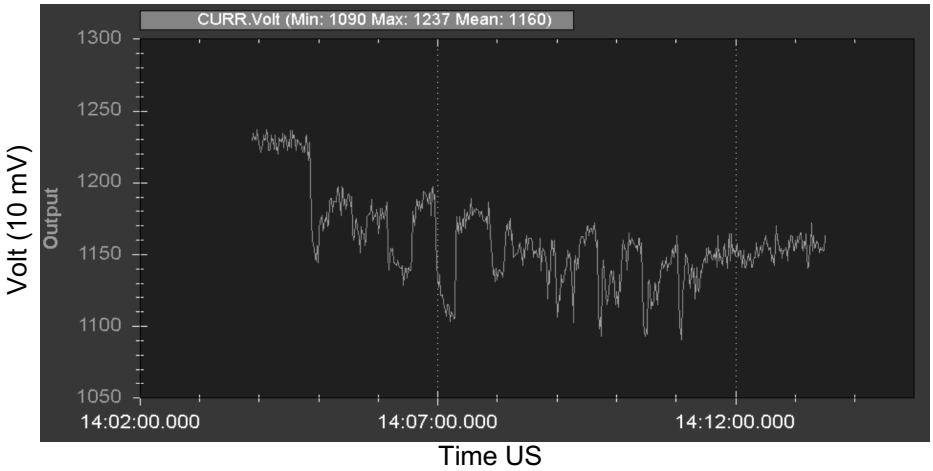


Fig. 20. Battery voltage level over a complete test flight.

4.10 Servos

The 13 g digital micro servo and 37 g standard servo shown in Fig. 21 were recommended by the airframe manufacturer. Two Eflr7150 servos were used for the ailerons. With a single Eflr7155 each for elevator and rudder.

Fig. 21. Airplane rudder, aileron, and elevator servos.

4.11 Radio Control Receiver

The Radio Control receiver by Hitec in Fig. 22 was used to control the airplane in “manual flight mode.” When desired, it was possible for the pilot to switch airplane control to direct (pilot) control via PWM signals using a channel in the receiver that controls the Servos Multiplexer.

Fig. 22. Radio control FM receiver.

4.12 Radio Control Transmitter

Fig. 23 shows the radio control transmitter by Futaba. This transmitter was used to fly the airplane in manual mode. It was used to override the autonomous flight if desired in any stage by switching to the dedicated channel “manual flight control” shown in Fig. 26. This could result in the “manual flight mux” to re-routing the path between auto-pilot controlling the servos and manual flight mode where the servos are controlled directly by the manual receiver.

Fig. 23. Radio control transmitter.

This feature was critical for two reasons. First, a pilot was required to take off and transition manually to auto-pilot. Second, in the event of an auto-pilot malfunctions, if a manual intervention by the pilot was required.

4.13 Hardware Integration

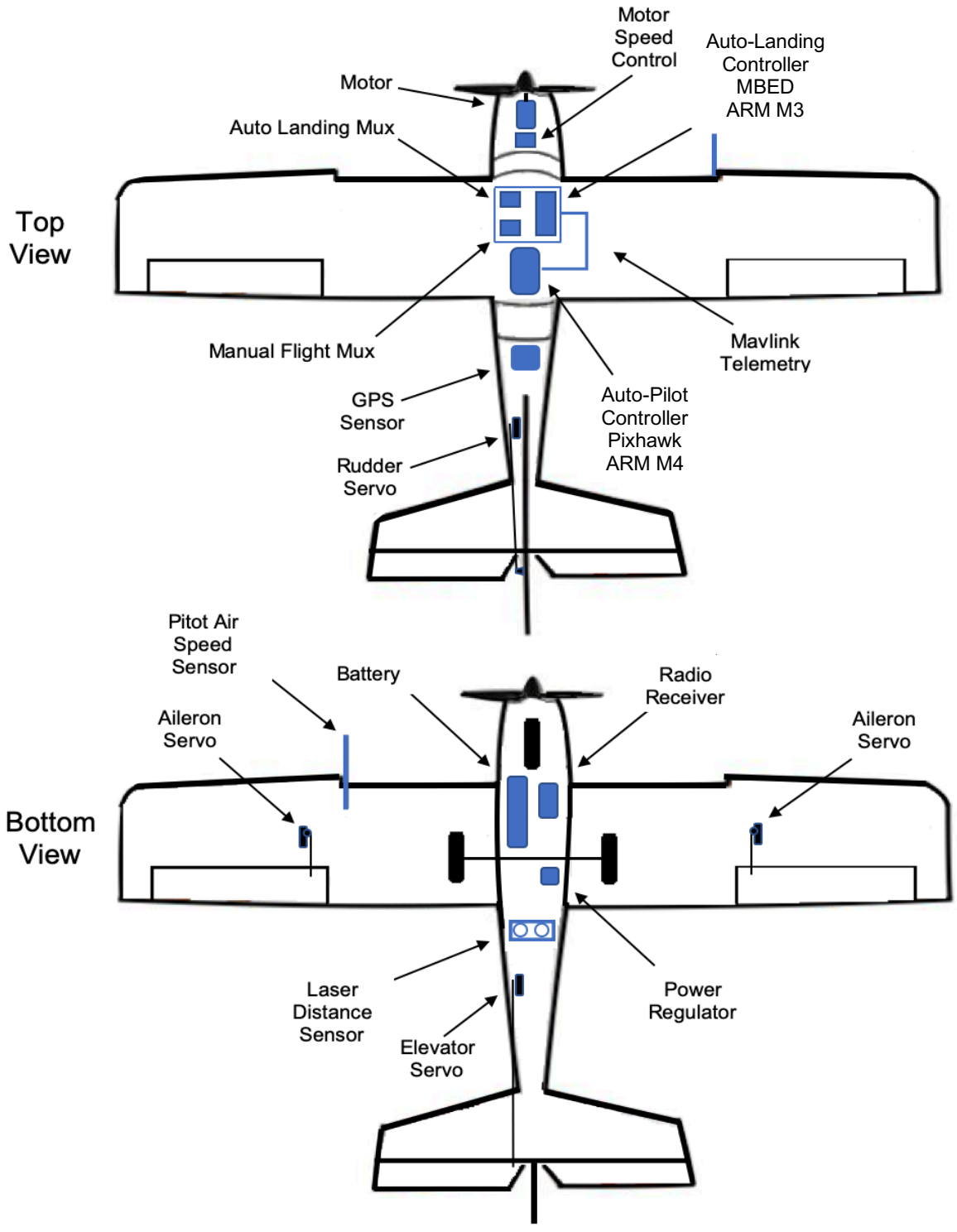


Fig. 24. Component locations within the airframe.

Integration and connectivity of this system as shown in Fig. 24 was challenging due to space limitations within the small airframe payload compartment. Placement of components inside the airplane had to be carefully planned as the space was limited and restrictions such as accessibility, heat, and weight distribution had to be considered. In order to have easier access to the internal components, external USB ports were installed in the aircraft to access the auto-landing MBED firmware and the Auto-Pilot Pixhawk controls. This enabled access to the internal equipment for firmware updates, control, and diagnostics as shown in the Fig. 25.

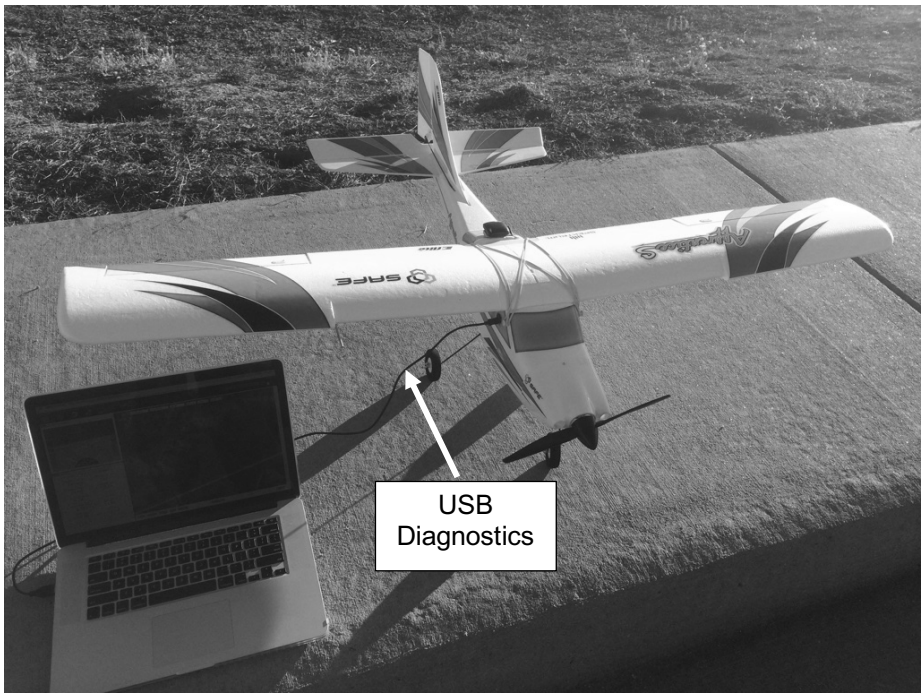


Fig. 25. USB diagnostics and control connection.

Other considerations for hardware placement were venting and cooling. Components such as CPUs, speed controllers, power regulators, batteries and

motors can generate heat and be affected by excessive heat. Heat sensitive components had to be placed in areas with higher air flow, while not affecting weight distribution. The airplane center of gravity was designed by the manufacturer. Changing the recommended center of gravity could cause the airplane to perform poorly.

The number of connections and wires in such a system can quickly overwhelm the available payload capacity. The requirement was to use a reliable common digital bus between the auto-pilot and the sensors. The decision was to use I2C, one of the available options on the Pixhawk. It used only 4 wires and was compatible with all the peripherals in this system. This solution helped minimize the number of wires and connectors. The I2C bus can operate at 100 Khz or 400 Khz. Both speeds exceeded the data rates required by the hardware in this system. Fig. 26 demonstrates the connection architecture by airplane control mode priority:

1. Operator – pilot can take control over the airplane using the “Manual Flight MUX.” This Multiplexer overrides any other controller.
2. Auto-Landing Controller –ARM M3 CPU takes control over the airplane using the “Auto Landing MUX.”
3. Auto-Pilot – Has control by default if 1 or 2 are disabled.

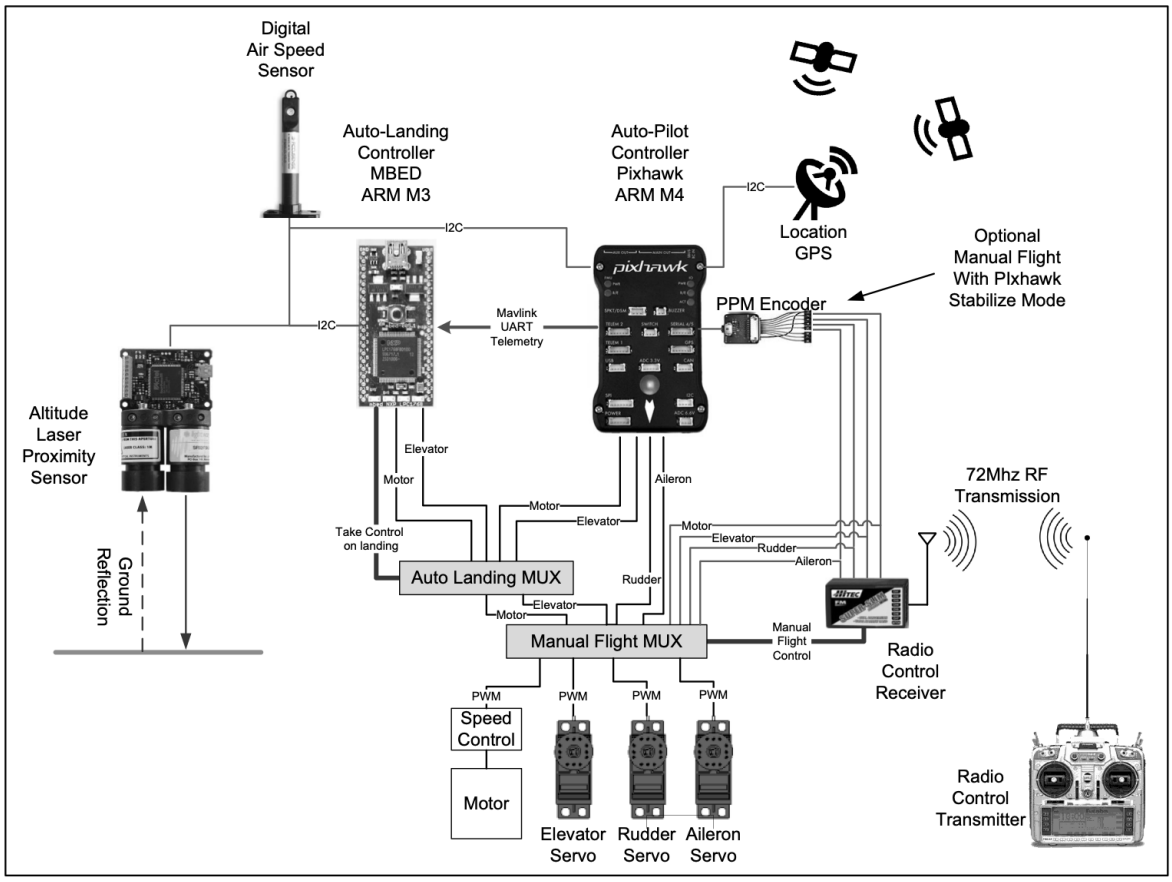


Fig. 26. Hardware placement and integration diagram.

This system used multiple voltage regulators to create the various voltages required by components. Fig. 27 demonstrates how the servos and receiver each required 4.8-volt inputs, and both the Pixhawk auto-pilot and I2C bus required a 5-volt input. The motor was powered by a 11.1 V lithium polymer battery via an ESC.

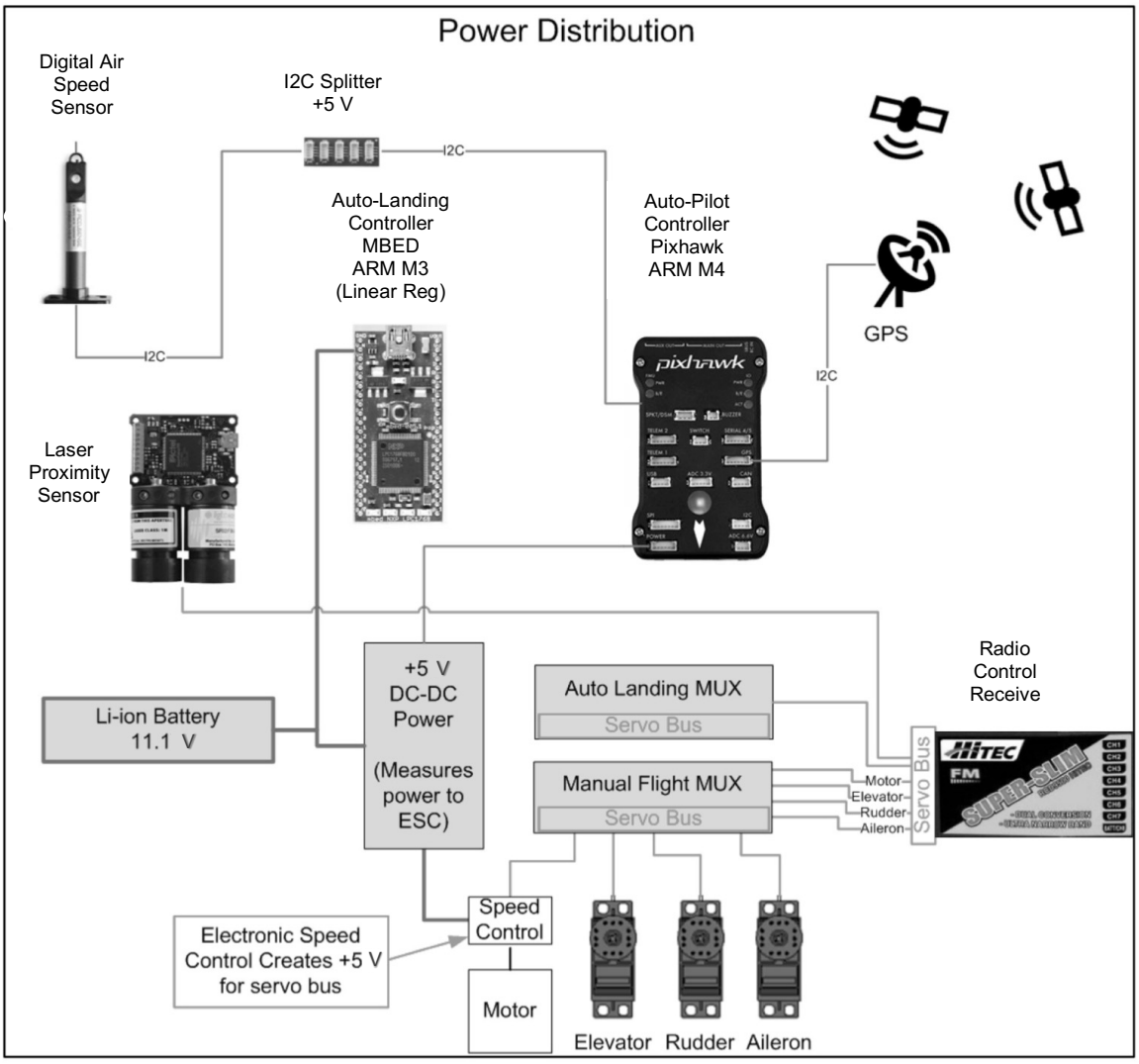


Fig. 27. Power distribution diagram.

4.14 Initial Hardware Testing

System integration and test began in an electronics lab. Fig. 28 show how all components were connected electrically on a breadboard. Basic firmware was running on the MBED ARM Cortex M3 (auto-landing Controller). This firmware was developed to test the connectivity of the auto-pilot via the auto-pilot's communication protocol, known as MavLink. The MavLink driver package was

installed on the MBED, and the initial telemetry was read and analyzed. An HD44780 LCD C++ driver was developed as a visual readout to aid with the debug process.

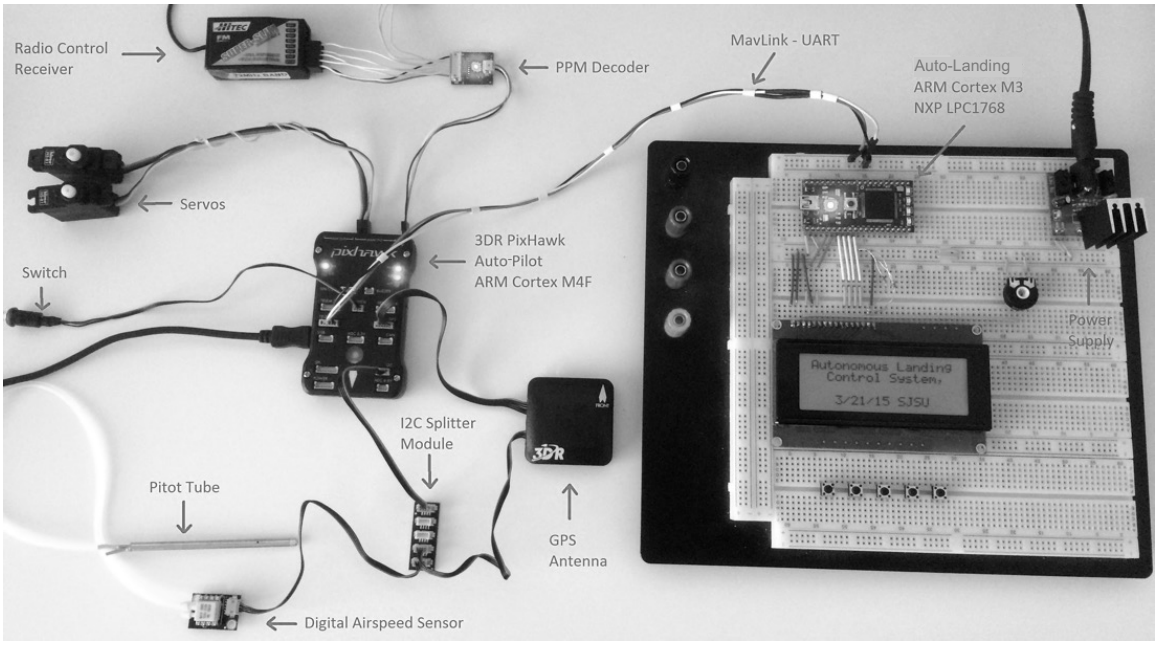


Fig. 28. Controllers communication via MavLink/UART serial.

4.15 MavLink Connectivity Testing

An oscilloscope was used to identify data packets and further analyze the MavLink serial interface. In Fig. 29, the auto-landing CPU communicated via MavLink on RS232/USB to a laptop running a Software in the Loop (SITL) simulator. The SITL simulated aircraft dynamics and auto-pilot response and transmits telemetry to the auto-landing. The auto-landing then reacted by moving motors to align with the flight path of the simulator.

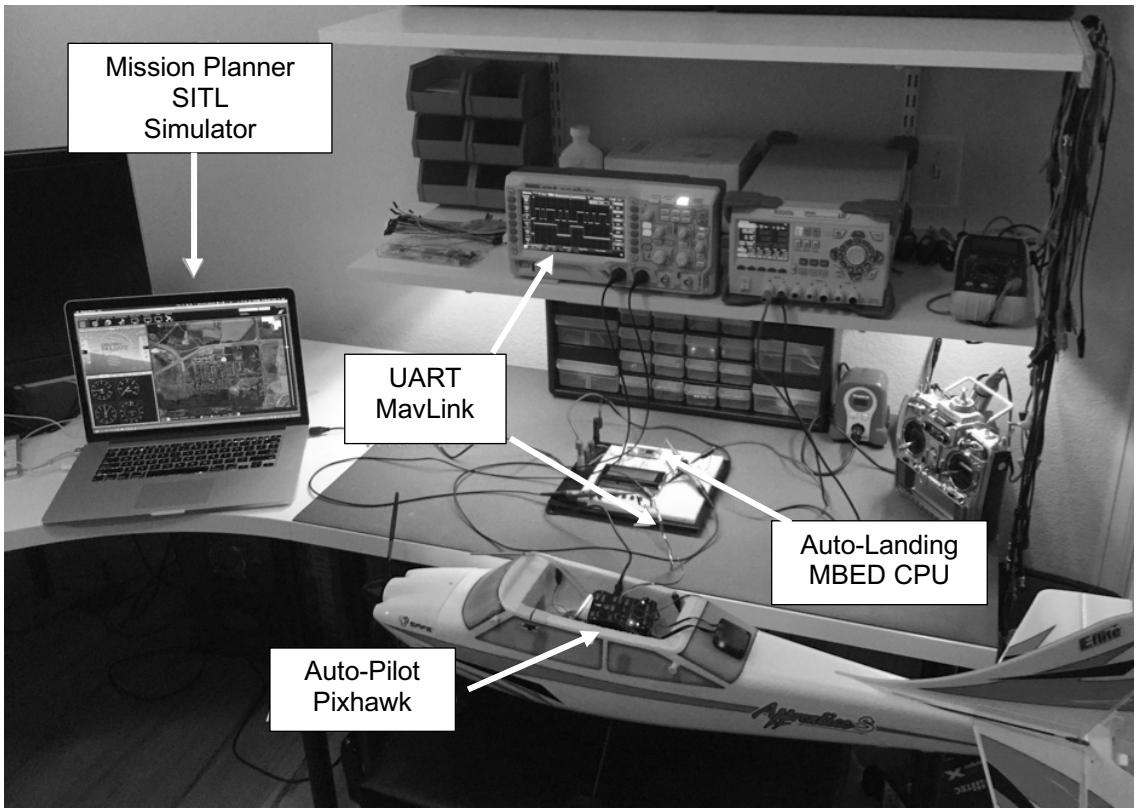


Fig. 29. Lab testing of MavLink bus.

Serial traffic of packets from the SITL ArduPlane simulator via MavLink are shown in Fig. 30. This test was used in the early stages of integration to debug an issue with the auto-landing controller because it was not receiving or recognizing MavLink telemetry packets. The oscilloscope helped to inspect the health of the physical protocol layer. This inspection exonerated the auto-landing controller.

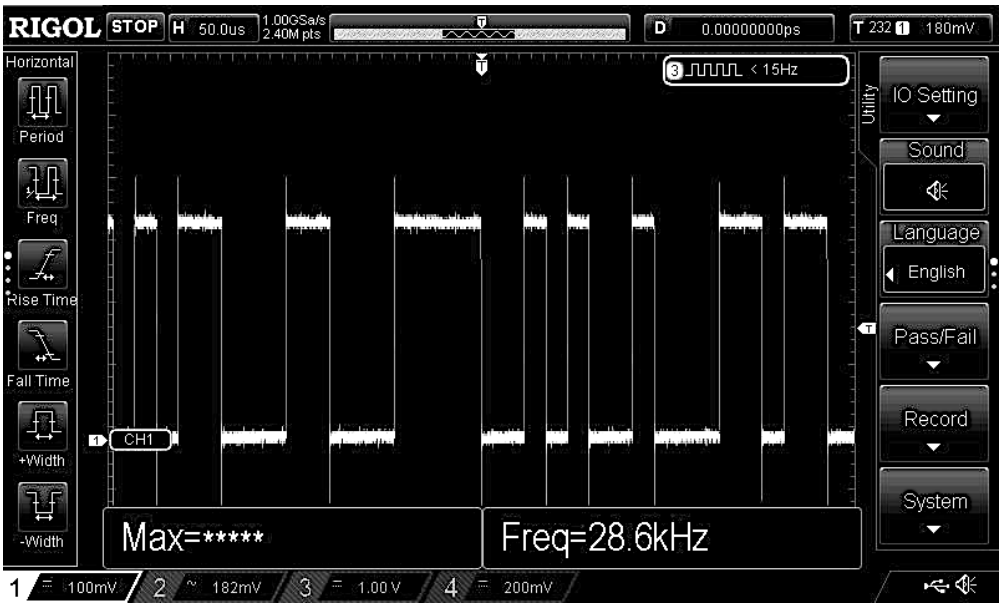


Fig. 30. MavLink serial traffic.

4.16 Airframe Testing

System flight testing was performed at the Radio Control Airfield at the Reservoir in Boulder, CO starting on May 21, 2015. Multiple flight tests were conducted in order to ensure the aircraft operated as expected with the payload weight of sensors and electronics. Battery capacity limited the duration of the test flights to between 10 to 20. The initial test flights were used to ensure that the platform was capable of flying safely with the added payload and sensors and that the aerodynamics of the airframe were preserved. In later stages of the project, the test flights shifted focus to collecting telemetry to gather enough data for the development of an accurate airplane transfer function. One of the test flights took place while the weather conditions were not ideal. During a take-off attempt in a high cross wind, the airplane tipped over and damaged the propeller.

Regardless of the damage and wind, it was decided to continue the test flight. The airplane as shown in Fig. 31 was still able to perform stable flight and followed the pre-programmed waypoints. This exercise proved the robustness of this system.

Fig. 31. Fixed-wing system during flight test.

4.17 Airplane Hardware Integration

4.17.1 Auto-Landing Controller and Multiplexer Board

Fig. 32 shows the early development stages of a small board that was used to host the auto-landing MBED NXP Cortex M3 controller and both multiplexers. Hardware integration and auto-pilot programming began on August 2, 2015.

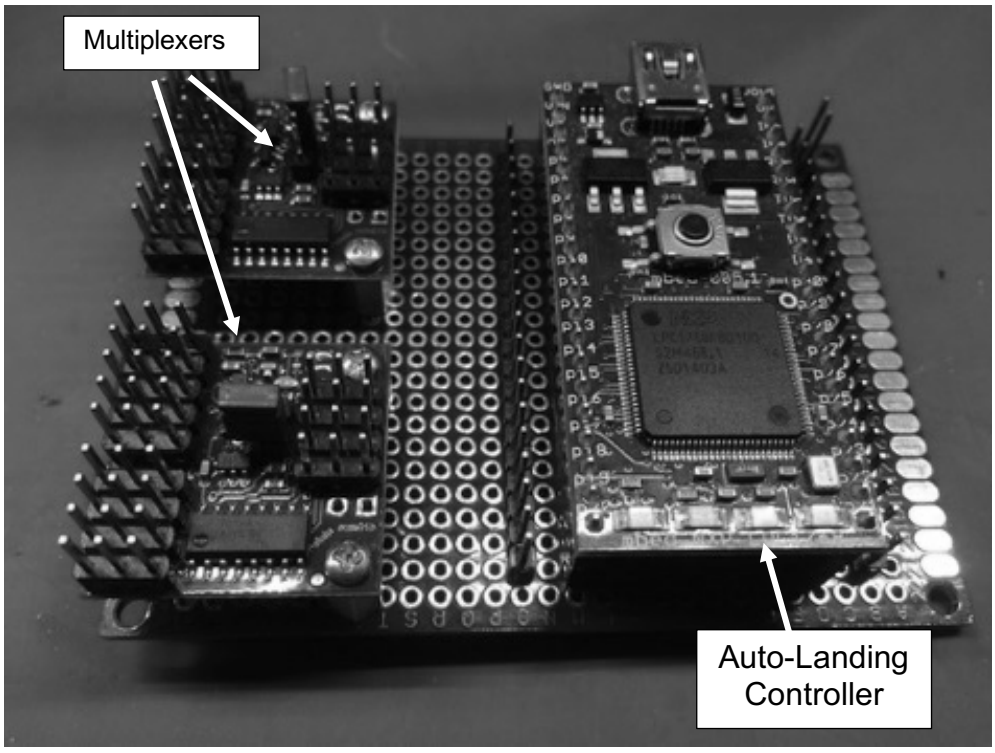


Fig. 32. Auto-landing controller and multiplexer board.

4.17.2 Payload Hardware Configuration

Fig. 33 demonstrates the available payload space inside the airplane. This aircraft platform was not originally designed to host all of the electronics that were used in this project. Creative methods had to be utilized to fit everything inside the volume of only 8 cm x 19 cm x 7 cm. Components were placed in two layers to accommodate the compartment shape. The wires were secured to prevent disconnections of critical systems resulting from motor or propeller vibrations.

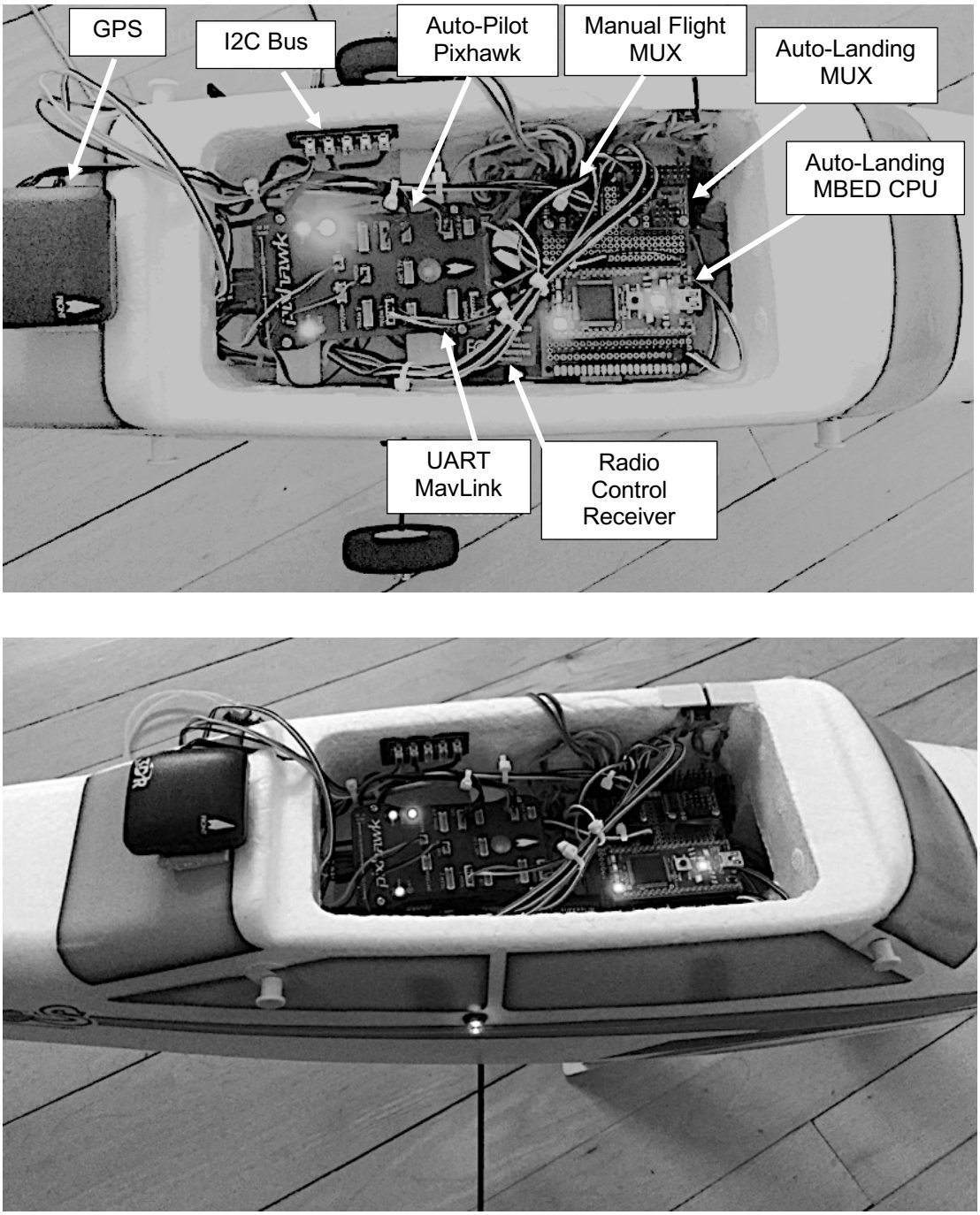


Fig. 33. Populated payload compartment.

5 SOFTWARE

5.1 Simulation Environment

The sophisticated software environment used in this project accommodated a few software simulators running simultaneously, communicating with the airplane auto-pilot and the auto-landing microcontroller. This environment was capable of emulating flight in order to exercise the airplane onboard hardware by sending simulated telemetry and replacing the onboard sensors data.

Additionally, this environment enabled simulation using the actual flight waypoints. Fig. 34 shows the actual waypoint file used in the test flight and simulations. The file describes GPS coordinates, altitude (120 feet in this case) and speed. These same waypoints were used repeatedly in the test flights as most of them were performed at the same airfield. Additional details on waypoint are discussed in the “Waypoint Programming” Section.

```
QGC WPL 110
0       1       0       16      0       0       0       0       40.085781       -105.233047     1593.000000     1
1       0       3       16      0.000000        0.000000        0.000000        0.000000        40.087215       -105.232353     120.000000      1
2       0       3       16      0.000000        0.000000        0.000000        0.000000        40.083908       -105.232910     120.000000      1
3       0       3       16      0.000000        0.000000        0.000000        0.000000        40.083759       -105.231163     120.000000      1
4       0       3       16      0.000000        0.000000        0.000000        0.000000        40.087067       -105.230583     120.000000      1
5       0       3       177     1.000000        8.000000        0.000000        0.000000        0.000000        0.000000        0.000000        1
```

Fig. 34. Waypoint file content.

The MavLink telemetry data were read and verified by the auto-landing controller. After reading the MavLink data the auto-landing controller managed the throttle and elevator, as expected, by switching the bypass multiplexer and directly driving the elevator and throttle servos. The system diagram is shown in Fig. 35.

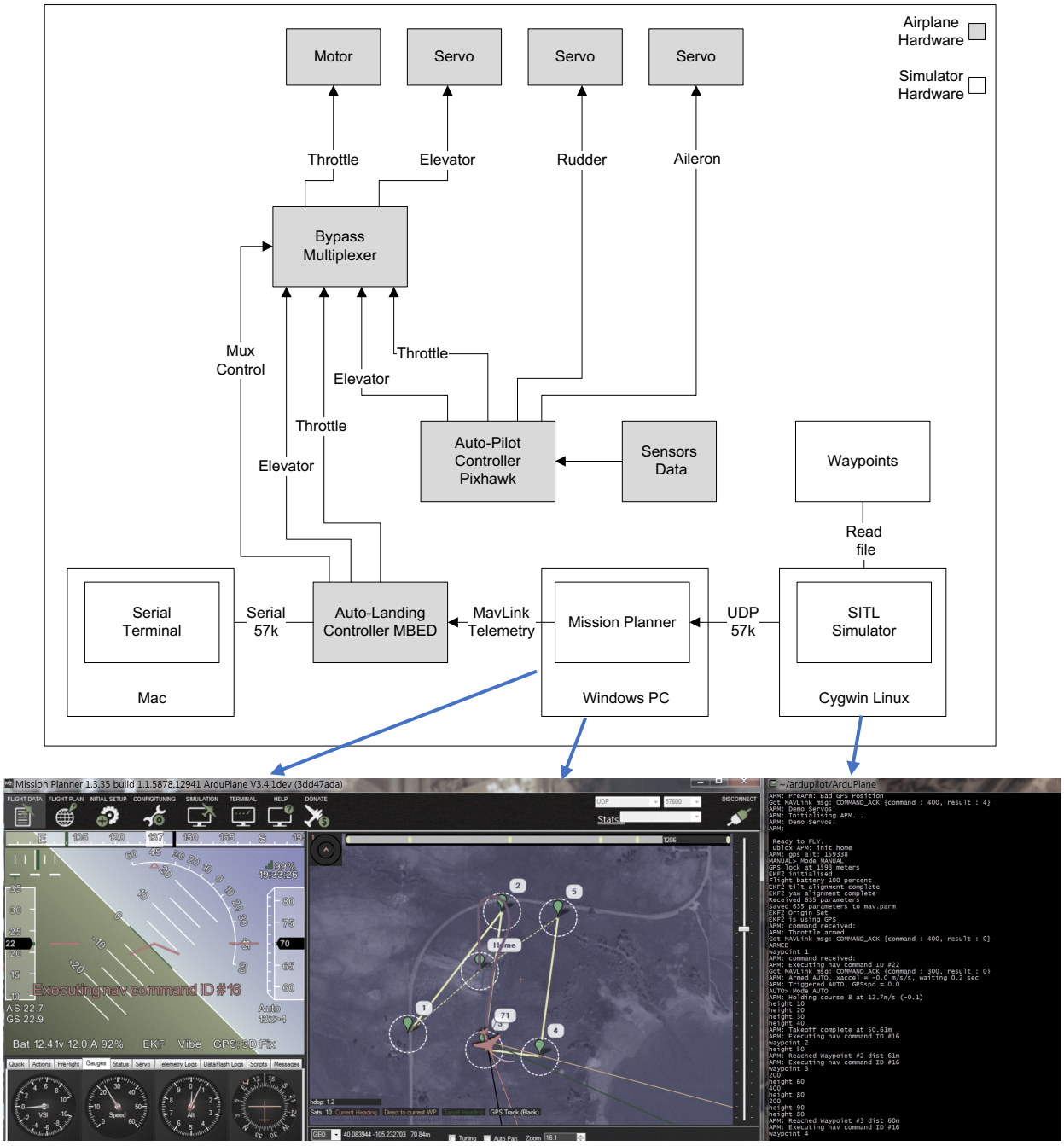


Fig. 35. Simulator integration with hardware.

An additional serial interface between the auto-landing and an external Macintosh computer was added to demonstrate how MavLink packets are being

processed by the firmware. Fig. 36 shows several packets with their corresponding packet ID. Each packet extracted data such as laser sensor altitude, GPS altitude, and air speed.

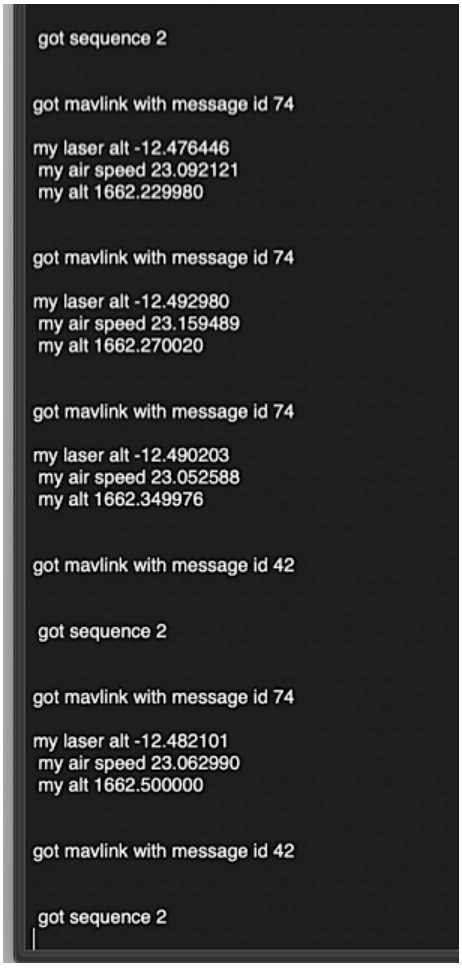


Fig. 36. Telemetry data via MavLink on PC terminal.

5.1.1 ArduPilot

ArduPilot is an open source auto-pilot software suite for fixed wing aircraft. This suite consists of navigation software and is compatible with a variety of embedded hardware platforms. ArduPilot provides the following features:

autonomous flight mode with waypoints, stabilization options, simulation including the ArduPilot SITL, support of navigation sensors, and numerous bus protocols to support sensor communication. In this thesis, ArduPilot was used in conjunction with SITL.

5.1.2 SITL Simulator (Software in the Loop)

The SITL simulator allows emulation of ArduPilot directly on a PC. SITL takes advantage of ArduPilot portability and compatibility and even enables the simulation of a flight following re-programmed waypoints, a critical capability in the development of the airplane hardware integration. SITL running ArduPilot simulation enabled significant testing to occur in the lab instead of the field by bypassing the Pixhawk auto-pilot hardware and connecting of the telemetry MavLink directly from the PC to the auto-landing controller. In Fig. 37 SITL simulator was ARMED and ready to fly.

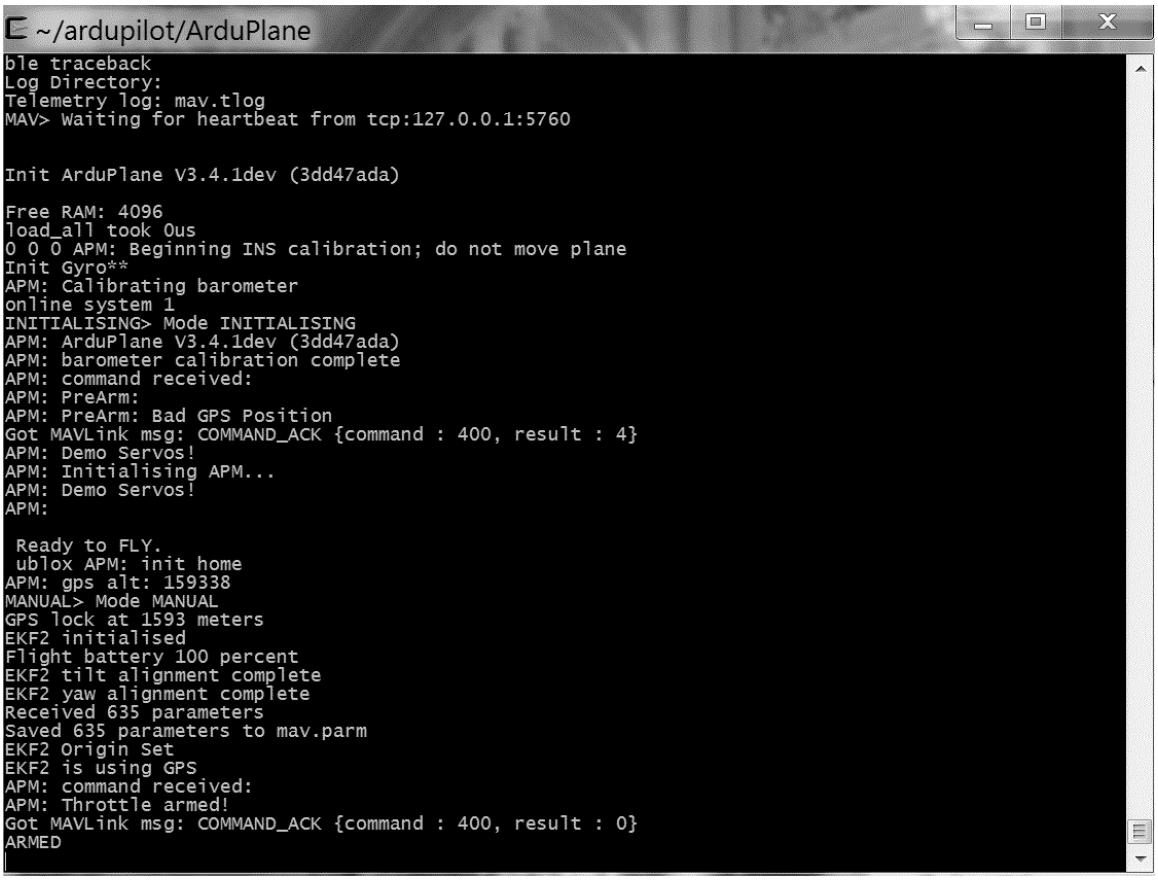


Fig. 37. SITL while executing a simulation.

5.1.3 Mission Planner

Mission Planner can be used to plan a flight, simulate a flight in combination with SITL, download mission log files, convert log files to MATLAB format, analyze log files, and configure the auto-pilot. In Fig. 38 Mission Planner shows a simulated flight over the Boulder, CO airfield, where the flight experiments took place using the exact same waypoints.

Fig. 38. Mission planner executing a simulation over the boulder airfield.

5.2 Pixhawk

5.2.1 ArduPlane

The Pixhawk auto-pilot chosen as part of the system configuration uses an ArduPlane firmware under development since 2010. This firmware enables the airplane to fly autonomously using the Pixhawk sensors and preprogrammed waypoints. This component configuration was ideal to preserve available scope for the vehicle landing algorithm development.

5.2.2 Waypoint Programming

A waypoint file was generated that contains GPS coordinates and altitude, airspeed, etc. In the example below the altitude is 70 feet. This waypoint file was used as inputs to program the auto-pilot with the desired flight path. The same waypoint file was loaded into the SITL simulator in order to simulate the same

flight on the computer. Fig. 39 and Fig. 40 demonstrate the simulation utilizing the waypoint file to fly the plane from waypoint 3 to 4.

```
QGC WPL 110
0       1       0       16      0       0       0       0       40.085781       -105.233047     1593.000000     1
1       0       3       16      0.000000        0.000000        0.000000        0.000000        40.087215       -105.232353     70.000000       1
2       0       3       16      0.000000        0.000000        0.000000        0.000000        40.083908       -105.232910     70.000000       1
3       0       3       16      0.000000        0.000000        0.000000        0.000000        40.083759       -105.231163     70.000000       1
4       0       3       16      0.000000        0.000000        0.000000        0.000000        40.087067       -105.230583     70.000000       1
5       0       3       177     1.000000        8.000000        0.000000        0.000000        0.000000        0.000000        0.000000        1
```

Fig. 39. Waypoint input file into SITL simulator.

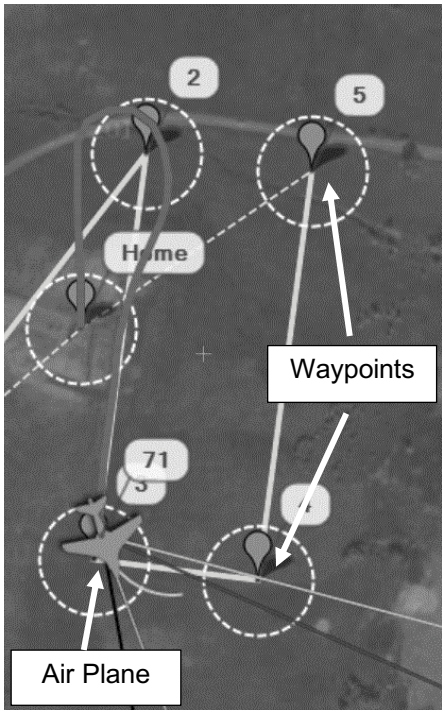


Fig. 40. Simulation on mission planner flying from waypoint 3 to 4.

5.2.3 Flight Log

All test flights were logged using the Pixhawk internal memory. The flight logs recorded all the sensor data. Experiments showed that flight time of about ten-minutes used approximately 40 MB of Flash memory space. The SD memory card in the Pixhawk had a 4 GB capacity. This enabled plenty of space for log

files. Fig. 41 shows a flight path over the Boulder airfield runway. The graph on the left shows the altitude throughout the test flight. The picture on the right shows the airplane path over the airfield and runways.

Fig. 41. Flight log on mission planner.

5.3 MavLink and Telemetry

Pixhawk provides a UART serial interface with a MavLink protocol. It was necessary to develop a driver to interface the auto-landing MBED controller (NXP LPC1768) with the Pixhawk auto-pilot for the transmission of telemetry to the auto-pilot for the control of flights mode, servos, etc. The driver was based on an existing Mavlink library named mavlink.h. By including this library in the auto-landing MBED environment, the MBED program was able to read Mavlink messages as shown in this short example:

Example MavLink Code

```
#include "mavlink.h"
mavlink_message_t msg;
float my_air_speed, my_alt, my_laser_alt;
// Wait for the correct telemetry message; ID = 74 in this case
if(msg.msgid == 74){
    // Read laser sensor altitude
    my_laser_alt = mavlink_msg_altitude_get_bottom_clearance(&msg);
    // Read GPS sensor altitude
    my_alt = mavlink_msg_vfr_hud_get_alt(&msg);
    // Read air speed sensor
    my_air_speed = mavlink_msg_vfr_hud_get_airspeed(&msg);
}
```

Fig. 42 shows a typical MavLink software structure and the integration with hardware such as Pixhawk auto-pilot.

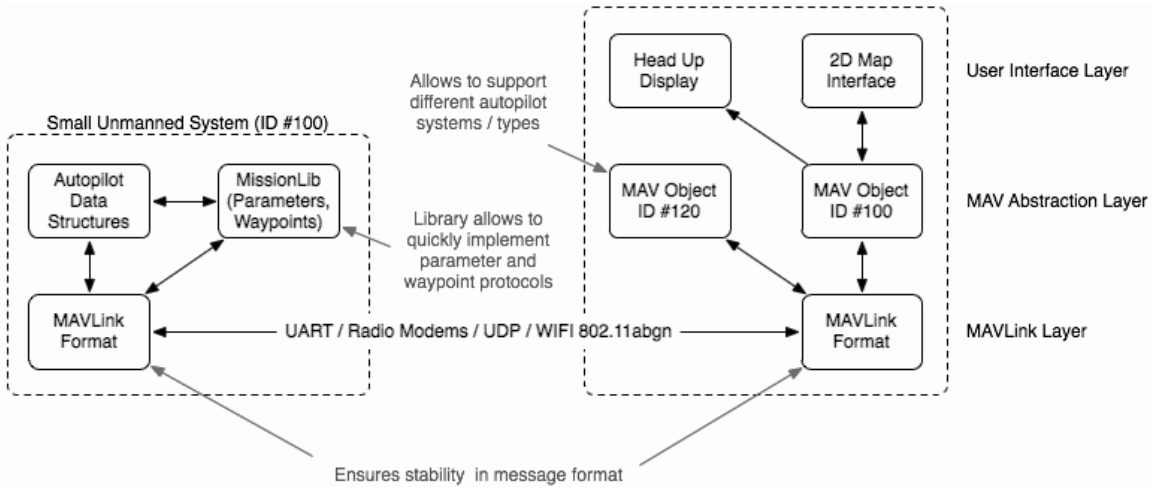
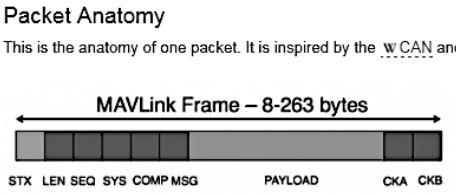


Fig. 42. Typical MavLink integration schematic [20].

Fig. 43 shows a Mavlink packet structure. Mavlink is a dedicated protocol for UAV communications. It is a serial interface with a header only message. This protocol uses CRC to ensure integrity of the messages. Each Mavlink message is identifiable by using an ID field. The maximum packet length is 263 bytes.



Byte Index	Content	Value	Explanation
0	Packet start sign	v1.0: 0xFE (v0.9: 0x55)	Indicates the start of a new packet.
1	Payload length	0 - 255	Indicates length of the following payload.
2	Packet sequence	0 - 255	Each component counts up his send sequence. Allows to detect packet loss
3	System ID	1 - 255	ID of the SENDING system. Allows to differentiate different MAVs on the same network.
4	Component ID	0 - 255	ID of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
5	Message ID	0 - 255	ID of the message - the id defines what the payload "means" and how it should be correctly decoded.
6 to (n+6)	Data	(0 - 255) bytes	Data of the message, depends on the message id.
(n+7) to (n+8)	Checksum (low byte, high byte)	ITU X.25/SAE AS-4 hash, **excluding packet start sign, so bytes 1..(n+6)** Note: The checksum also includes MAVLINK_CRC_EXTRA (Number computed from message fields. Protects the packet from decoding a different version of the same packet but with different variables).	

Fig. 43. MavLink packet structure [20].

5.4 Auto-Landing Controller Platform Software

C++ language was used in the development of the MBED auto-landing controller firmware. The base classes used stemmed from the MBED library, self-development, and MavLink library. MBED is a platform that was developed for

the NXP Cortex CPU. Utilizing the above classes allowed for a higher level and less complex programing approach. The available MavLink C++ library made it possible to more easily embed it into the MBED environment. LPCXpresso for Mac was the chosen development environment for the MBED controller firmware. MBED is a platform with its own online cloud environment, though an offline environment was utilized for this project.

6 CONTROLS RESEARCH AND DEVELOPMENT

6.1 Transfer Function Research and Development

To develop the descent algorithm, the transfer function of the fixed-wing airplane first needed definition. A transfer function is the key for any control loop. To find the transfer function experimentally, several simple flight maneuvers were conducted to reveal the response function of the airplane with the aim of acquiring the response of the airframe to a given control input. The data were recorded on an SD card hosted in the auto-pilot. The collected data showed input *vs*. output in the time domain.

6.2 Test Flights for Algorithm Data Collection

During the experimental flight shown in Fig. 44 and Fig. 45, the airplane was introduced to 12 “pull up” maneuvers using the elevator to simulate a step function. A step function as an input will make it possible to subsequently extract a transfer function.

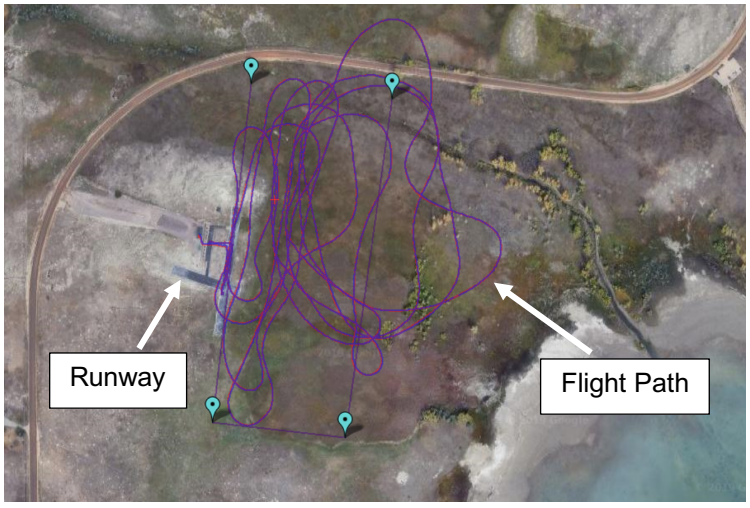


Fig. 44. Experimental flight near Boulder reservoir in Boulder, Colorado.

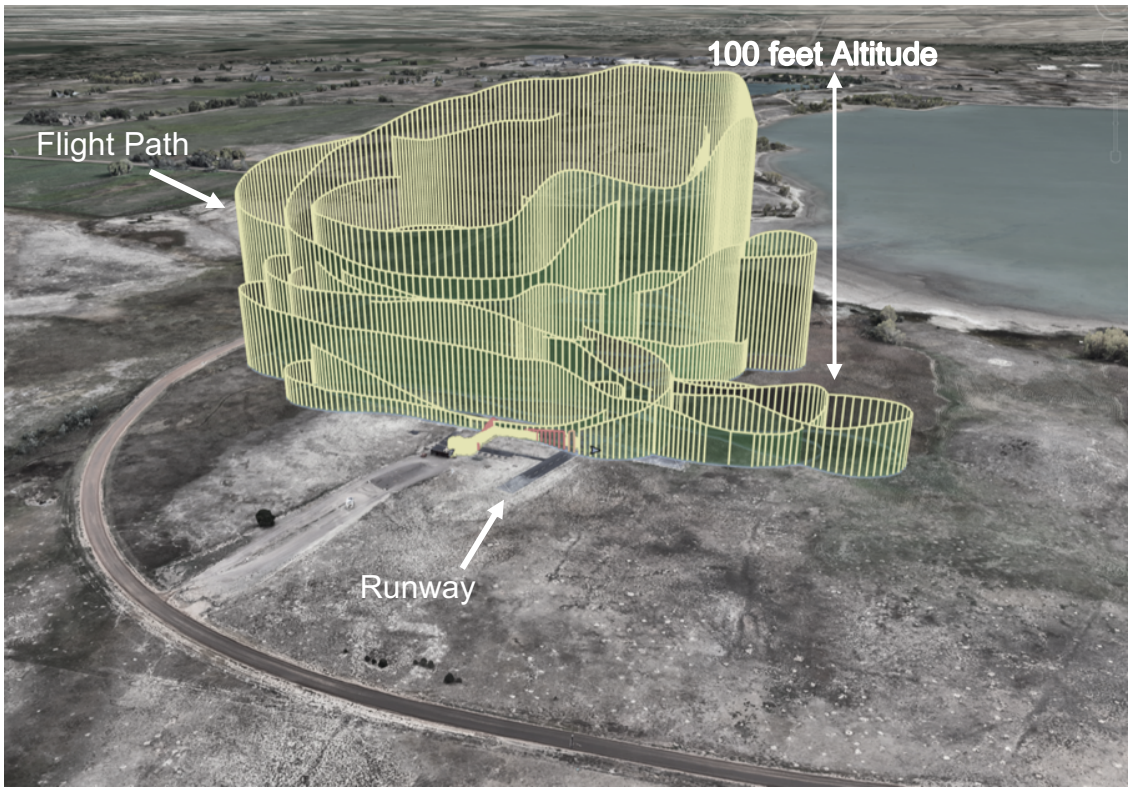


Fig. 45. Experimental flight taken Aug 2019 in Boulder, Colorado.

In order to import the data from the “Pixhawk Mission Planner” log file shown in Fig. 46, the data of the altitude and the elevator had to be manipulated and interpolated as the sampling rate and associated time stamps did not correlate. The elevator telemetry was collected at twice the rate of altitude data and at different times. As shown in Table 1, the time stamps of the elevator telemetry on the left do not match the time stamps of the altitude data on the right side.

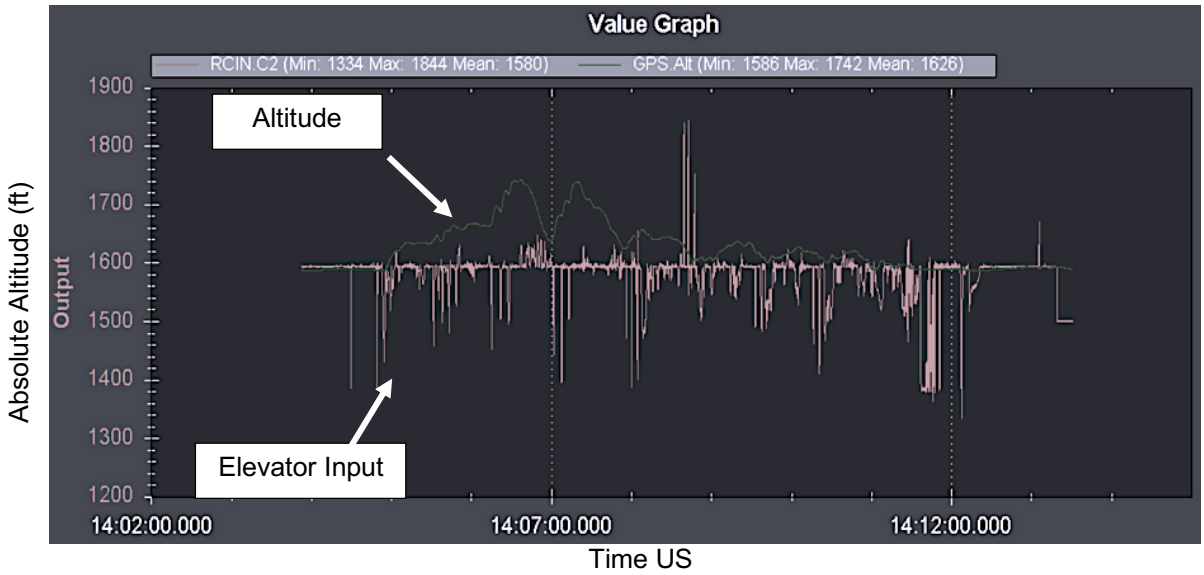


Fig. 46. Experiment flight data altitude *vs.* elevator input.

Table 1
Raw Flight Auto-Pilot Data

Elevator Time	Elevator PWM (ms)	Altitude (feet)	AltitudeTime
1346971439	1594	100.8	1346910227
1347070988	1591		
1347170809	1597	100.69	1347110433
1347270909	1594		
1347370824	1592	100.53	1347310463
1347471767	1589		
1347570980	1593	100.36	1347510262
1347670999	1584		
1347771066	1597	100.06	1347710871
1347871636	1589		
1347971657	1595	99.59	1347910418
1348070849	1585		
1348170929	1588	99.1	1348110648
1348271258	1596		
1348370849	1589	98.63	1348310431
1348472198	1592		
1348568218	1592	98.12	1348510349
1348671860	1596		
1348770691	1592	97.55	1348710267
1348871043	1587		
1348971426	1610	96.99	1348910603
1349071804	1592		
1349170787	1578	96.48	1349110122
1349271851	1592		
1349370871	1588	96.09	1349310138

Linear interpolation between two known data points was the method used to extract the missing data. In this method, the missing data point is retrieved by using the equation below to find a value on a straight line between two known data points.

$$\frac{y - y_0}{x - x_0} = \frac{y - y_0}{x - x_0}$$

When looking for value of y at point x:

$$y = \frac{y - y_0}{x - x_0}(x - x_0) + y_0$$

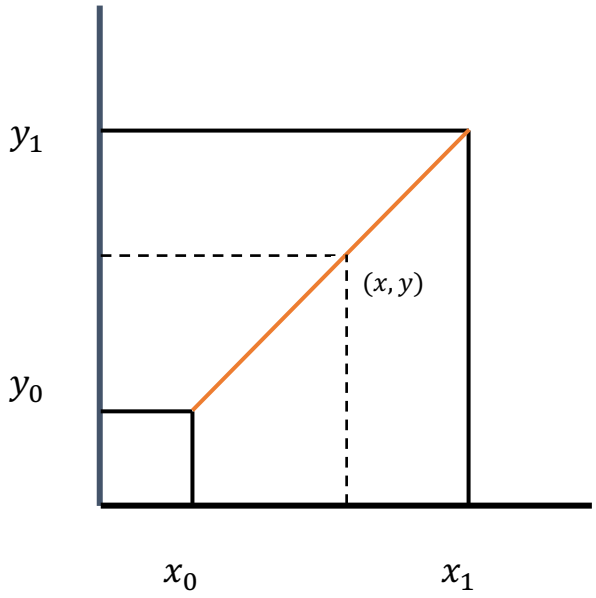


Fig. 47. Linear interpolation between two known points.

6.3 MATLAB Script and Data Interpolation

The script below was developed in order to match the sampling of the altitude and elevator data. The script used interpolation to fill the gaps of the missing data.

MATLAB Script

```
clc; clear; close all
set(gcf,'Visible','on')
RCINdata = load ('/flight_log.mat', 'RCIN')
GPSdata = load ('/flight_log.mat', 'GPS')
NumberOfRows = size(GPSdata.GPS,1);
for i = 2 : NumberOfRows
   Table(i,:)=    (find(RCINdata.RCIN(i,2) <= GPSdata.GPS(:,2),1));
   Table(i,2)=    RCINdata.RCIN(i,2);
   MatchedValue = (find(RCINdata.RCIN(i,2) <= GPSdata.GPS(:,2),1));
   if(MatchedValue > 1)
    Table(i,3)=    GPSdata.GPS(MatchedValue-1,2);
    Table(i,4)=    GPSdata.GPS(MatchedValue,2);
    Table(i,5)=    GPSdata.GPS(MatchedValue-1,10);
    Table(i,6)=    GPSdata.GPS(MatchedValue,10);
    Table(i,7)=    Table(i,5) + ((Table(i,6)- Table(i,5)) * (Table(i,2)-Table(i,3))/(Table(i,4)-Table(i,3)))
    Table(i,8)=    RCINdata.RCIN(i,4);
   end
end
```

After manipulating the data using the above script, the elevator (elevator PWM) and altitude (Interpolated Ralt) had the same sampling rate and time stamps.

Table 2
Matched Flight Auto-Pilot Data

Elevator Time	Elevator PWM (ms)	Altitude (feet)	Time(Pre)	Time(Post)	Value(Pre)	Value(Post)	Interpolated Ralt
1346971439	1594	100.8	1346910227	1347110433	100.8	100.69	100.766368
1347070988	1591	100.8	1346910227	1347110433	100.8	100.69	100.7116724
1347170809	1597	100.69	1347110433	1347310463	100.69	100.53	100.6417064
1347270909	1594	100.69	1347110433	1347310463	100.69	100.53	100.5616385
1347370824	1592	100.53	1347310463	1347510262	100.53	100.36	100.4786415
1347471767	1589	100.53	1347310463	1347510262	100.53	100.36	100.3927537
1347570980	1593	100.36	1347510262	1347710871	100.36	100.06	100.2691995
1347670999	1584	100.36	1347510262	1347710871	100.36	100.06	100.1196264
1347771066	1597	100.06	1347710871	1347910418	100.06	99.59	99.91822062
1347871636	1589	100.06	1347710871	1347910418	100.06	99.59	99.6813446
1347971657	1595	99.59	1347910418	1348110648	99.59	99.1	99.44013679
1348070849	1585	99.59	1347910418	1348110648	99.59	99.1	99.19739555
1348170929	1588	99.1	1348110648	1348310431	99.1	98.63	98.95818578
1348271258	1596	99.1	1348110648	1348310431	99.1	98.63	98.72215654
1348370849	1589	98.63	1348310431	1348510349	98.63	98.12	98.47587091
1348472198	1592	98.63	1348310431	1348510349	98.63	98.12	98.21732495
1348568218	1592	98.12	1348510349	1348710267	98.12	97.55	97.9550057
1348671860	1596	98.12	1348510349	1348710267	98.12	97.55	97.65950485
1348770691	1592	97.55	1348710267	1348910603	97.55	96.99	97.38109656
1348871043	1587	97.55	1348710267	1348910603	97.55	96.99	97.10058222
1348971426	1610	96.99	1348910603	1349110122	96.99	96.48	96.83452744
1349071804	1592	96.99	1348910603	1349110122	96.99	96.48	96.57794646
1349170787	1578	96.48	1349110122	1349310138	96.48	96.09	96.36171271
1349271851	1592	96.48	1349110122	1349310138	96.48	96.09	96.16465368
1349370871	1588	96.09	1349310138	1349510231	96.09	95.8	96.00197808

The corresponding graphical representation in Fig. 48:

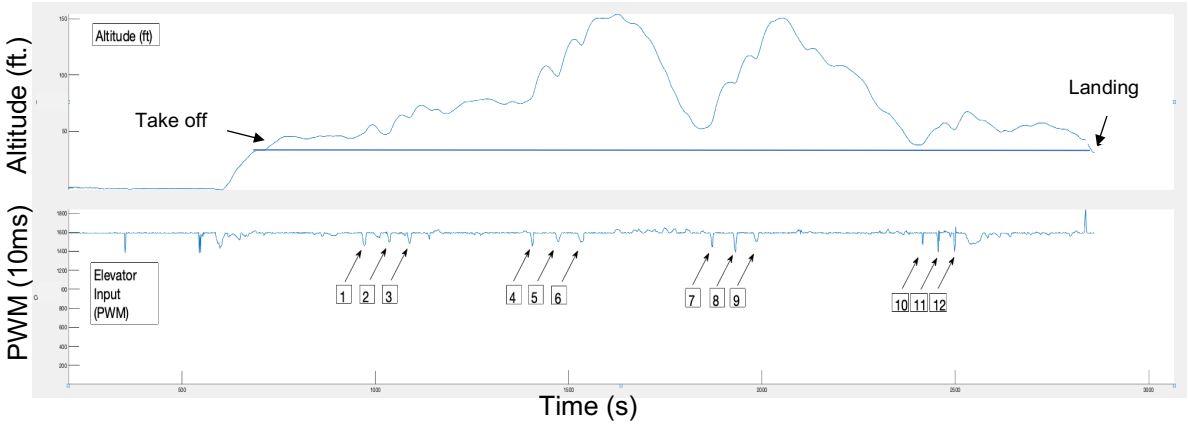


Fig. 48. Experiment flight data imported to MATLAB.

6.4 Analysis of Logged Flight Data

After examination of the flight data, the data below were used to extract the transfer function.

Table 3
Sampled Auto-Pilot Data

Elevator Time	Time(Pre)	Time(Post)	Value(Pre)	Value(Post)	Interpolated Ralt	Elevator PWM (ms)
146778252	146637991	146838091	47.14	47.47	47.37131499	1593
146877696	146838091	147038093	47.47	47.79	47.53336737	1570
146978174	146838091	147038093	47.47	47.79	47.69413056	1550
147075553	147038093	147257442	47.79	48.11	47.84464898	1519
147178589	147038093	147257442	47.79	48.11	47.99496433	1483
147278835	147257442	147438225	48.11	48.53	48.1597008	1464
147378321	147257442	147438225	48.11	48.53	48.39082939	1460
147477657	147438225	147657500	48.53	49.21	48.65228371	1458
147578052	147438225	147657500	48.53	49.21	48.96362153	1465
147678578	147657500	147838070	49.21	50.12	49.31622462	1472
147778133	147657500	147838070	49.21	50.12	49.81794168	1510
147877889	147838070	148038074	50.12	51.17	50.32904557	1550
147975449	147838070	148038074	50.12	51.17	50.84122533	1592
148077640	148038074	148238400	51.17	52.26	51.38528379	1595
148178006	148038074	148238400	51.17	52.26	51.93138834	1593
148277623	148238400	148438298	52.26	53.28	52.46013937	1595
148377982	148238400	148438298	52.26	53.28	52.97223144	1595
148477591	148438298	148638386	53.28	54.11	53.44299423	1595
148577832	148438298	148638386	53.28	54.11	53.85881142	1594
148677705	148638386	148838408	54.11	54.75	54.23580696	1595
148778239	148638386	148838408	54.11	54.75	54.55748038	1596
148877776	148838408	149057302	54.75	55.23	54.83632781	1595
148975474	148838408	149057302	54.75	55.23	55.05056411	1594
149079073	149057302	149238590	55.23	55.55	55.26842902	1594
149177978	149057302	149238590	55.23	55.55	55.4430109	1595
149277710	149238590	149438321	55.55	55.71	55.58133815	1594
149377807	149238590	149438321	55.55	55.71	55.6615236	1595
149477901	149438321	149638546	55.71	55.71	55.71	1595
149577986	149438321	149638546	55.71	55.71	55.71	1595
149677908	149638546	149857709	55.71	55.54	55.67946775	1592
149777914	149638546	149857709	55.71	55.54	55.60189526	1595
149879274	149857709	150057778	55.54	55.2	55.50335214	1594

Further inspection of the data as captured from the auto-pilot log showed portions of the data were too noisy and not in an ideal format for the MATLAB identification tool. Fig. 49 and Fig. 50 demonstrate the data before and after manipulations.

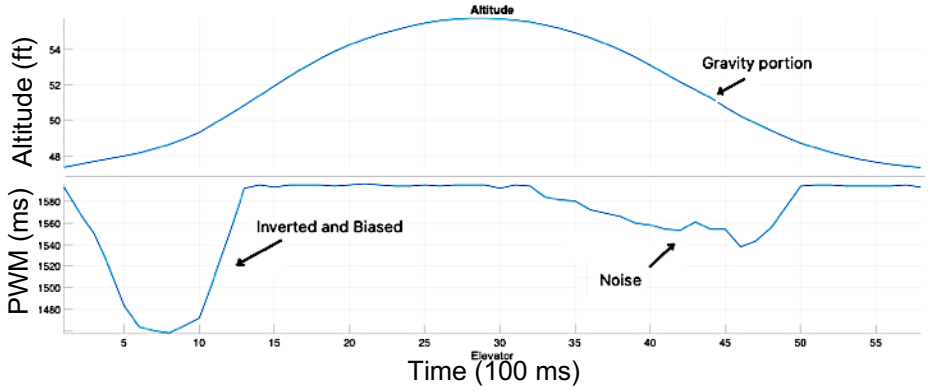


Fig. 49. Raw sampled log data (top) altitude, (bottom) elevator.

The noise portion was removed, because it did not represent optimal operation. The gravity portion was ignored as it is not part of the airplane response to the elevator input. The data were extended to 10 seconds to achieve better resolution in the frequency domain, as Frequency Resolution = sample rate/window size. The elevator was inverted and normalized to optimize the data to the identification tool. The result can be seen in Fig. 50.

Elevator_normalized = Pullup elevator x -1+1595’

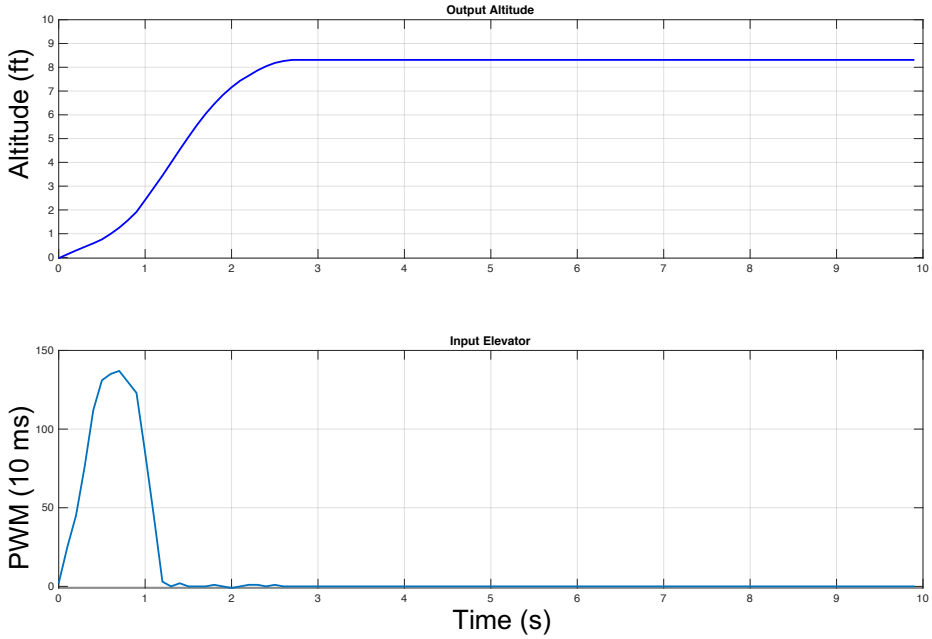


Fig. 50. Manipulated sampled log data.

6.5 Transfer Function Development

Transfer functions in control theory characterize the relationship between the input and output. Transfer functions can be represented in linear, fixed time, differential equations. Assume an input driving function and output response

function, while initial conditions are zero. The ratio of the Laplace transform of the output *vs.* the input is the definition of a transfer function.

The following linear fixed time system is represented by the following differential equation [21].

$$a_0{}^{(n)}y + a_1{}^{(n-1)}y + \cdots + a_{n-1}\dot{y} + a_n y$$

$$= b_0{}^{(m)}x + b_1{}^{(m-1)}x + \cdots + b_{m-1}\dot{x} + b_m x \quad (n \geq m)$$

when x is the input and y is the output. The ratio between the $y(s)$ Laplace transform to the $x(s)$ Laplace transform when input conditions are zero is the system transfer function.

$$\textit{Transfer function} = G(s) = \frac{\mathcal{L}[output]}{\mathcal{L}[input]}\Big|_{\textit{zero initial conditions}}$$

$$= \frac{Y(s)}{X(s)} = \frac{b_0 s^m + b_1 s^{m-1} + \cdots + b_{m-1}s + b_m}{a_0 s^n + a_1 s^{n-1} + \cdots + a_{n-1}s + a_n}$$

Transfer functions can define system dynamics using algebraic equation form in s. The system called nth order system when the denominator contains the highest power of s that equal to n [21].

Assuming a linear fixed time system *G(s)*. Shown in Fig. 51. The transfer function is

$$G(s) = \frac{Y(s)}{X(s)}$$

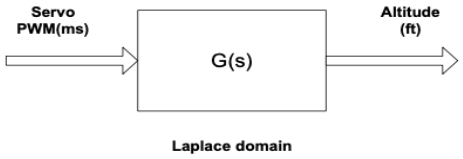


Fig. 51. Plant input and output.

6.6 System Identification in MATLAB Overview

Experimental data can be used to find a transfer function and the system dynamics. Software tools such as the MATLAB Identification Tool as seen in Fig. 52 can generate an estimated continuous-time transfer function based on experimental or available input and output data. A given system can be represented by the relationship of input to output. A linear system transfer function can be defined as the Laplace transform ratio of input to output. The MATLAB Identification tool generates a continuous-time transfer function for imported data arrays of input and output. Additional information such as sampling rate of the data and estimation of the numbers of poles and zeros are required for a successful result.

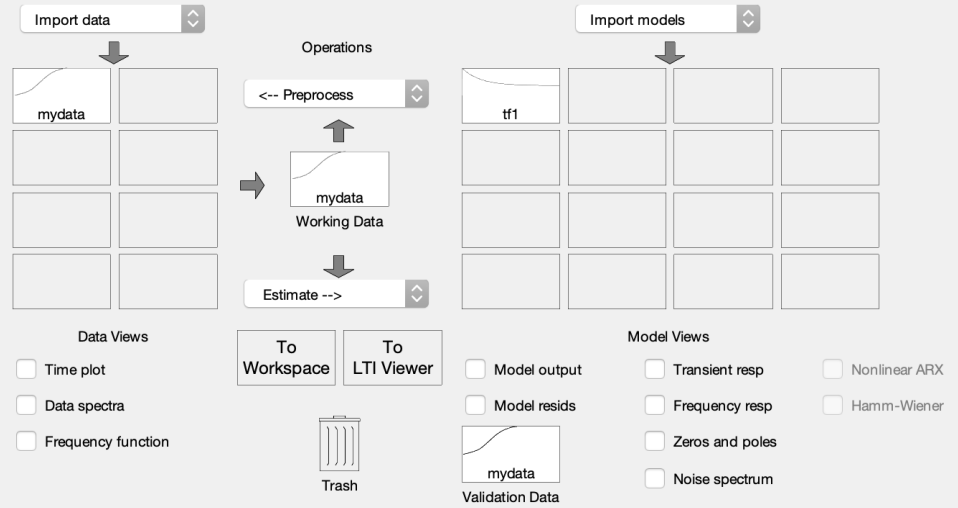


Fig. 52. System identification interface.

6.7 System Identification in MATLAB Output

The chosen experimental data had to be modified in order to run it through the identification tool. The elevator input was normalized, and the bias was removed and inverted, as shown in Fig. 53.

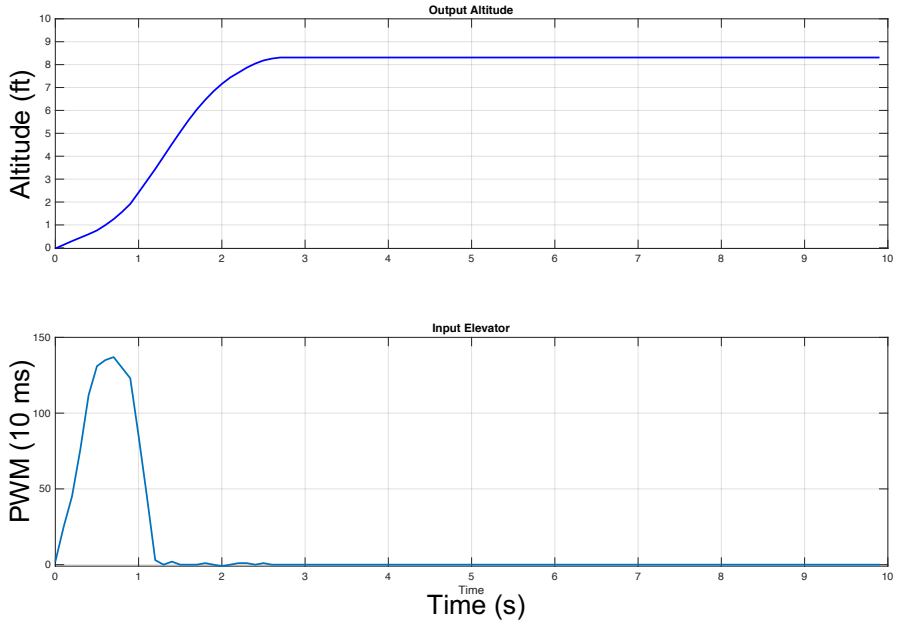


Fig. 53. System identification input.

```
Transfer Function Identification
Estimation data: Time domain data mydata
Data has 1 outputs, 1 inputs and 100 samples.
Number of poles: 2, Number of zeros: 1
Initialization Method: "iv"



Estimation Progress
Initializing model parameters...
Initializing using 'iv' method...
done.

Initialization complete.

Algorithm: Nonlinear least squares with automatically chosen line search method
-------------------------------------------------------------------------------------------
                          Norm of     First-order     Improvement (%)
Iteration      Cost       step        optimality    Expected  Achieved   Bisections
-------------------------------------------------------------------------------------------
    0     0.0289457          -      1.62e+03   1.82e+03          -        -
    1     0.0233931       9.17      7.82e+03   1.82e+03       19.2        2
    2     0.0205314       6.21      1.24e+04    1.7e+03       12.2        2
    3     0.0192387       8.73      2.12e+04   1.48e+03        6.3        1
    4     0.0167436       8.02      2.14e+04    1.3e+03         13        0
    5     0.0144572       0.65          95.2        816       13.7        0
    6     0.0144568     0.0363          3.68      0.206     0.00235        0
    7     0.0144568    0.00846         0.186    0.00992    0.000111        0
-------------------------------------------------------------------------------------------
Estimating parameter covariance...
done.

Result
Termination condition: Near (local) minimum, (norm(g) < tol)..
Number of iterations: 7, Number of function evaluations: 20

Status: Estimated using TFEST
Fit to estimation data: 95.05%, FPE: 0.0163024
```

```
  From input "u1" to output "y1":
    -0.03266 s + 0.09495
  ------------------------
  s^2 + 1.609 s + 0.006601
Name: tf1
Continuous-time identified transfer function.

Parameterization:
   Number of poles: 2   Number of zeros: 1
   Number of free coefficients: 4
   Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using TFEST on time domain data "mydata".
Fit to estimation data: 95.05% (stability enforced)
FPE: 0.0163, MSE: 0.01446
```

Fig. 54. MATLAB identification tool output report.

The airplane transfer function from the Identification Tool using the experiment Pullup1 data with a 95.05% fit:

$$TF(s) = \frac{-0.03266s + 0.09495}{s^2 + 1.609s + 0.006601}$$

The fit in Fig. 55 resulted in a 95% confidence of matching the actual modeled airplane. This result was sufficient for a control loop design. The frequency response is shown in Fig. 56. The system is linear as no delays were needed to be added.

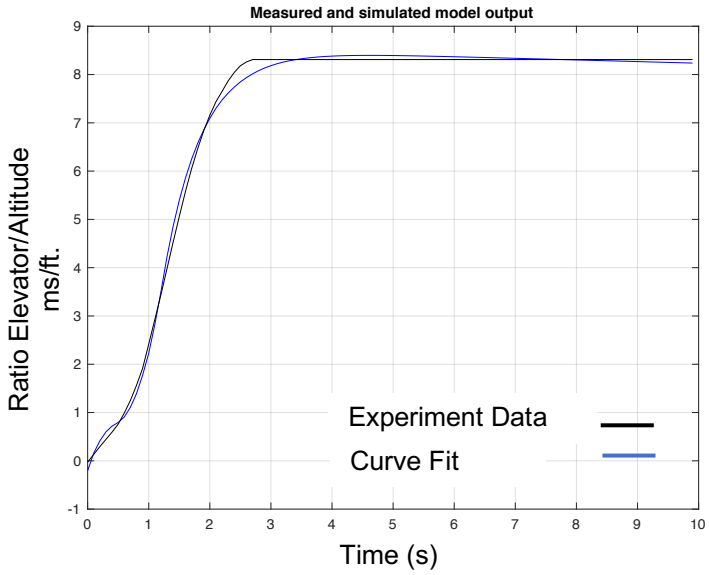


Fig. 55. Airplane transfer function curve fit accuracy.

Although a small delay would have been acceptable to achieve stability, a larger delay would have been more challenging to control. Fig. 56 demonstrates the Bode plot frequency response of the transfer function.

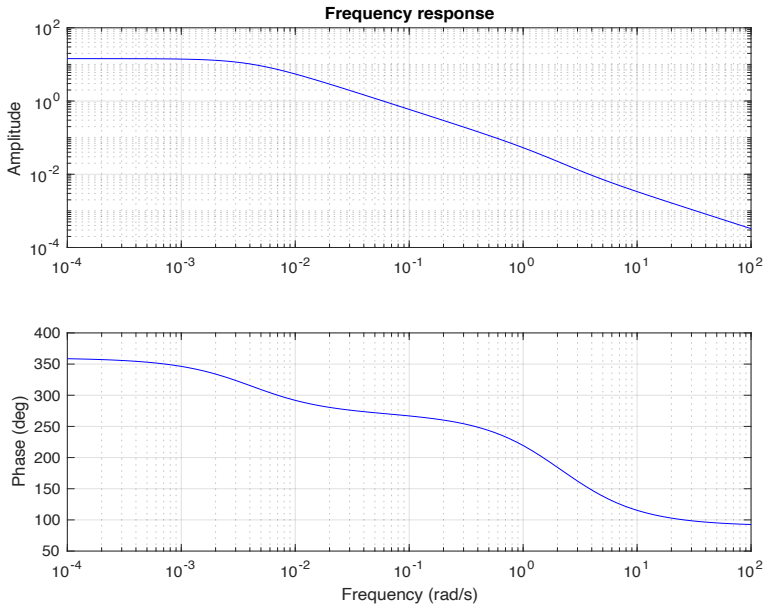


Fig. 56. Identification tool frequency response output.

6.8 Bode Plot

6.8.1 Bode Plot Analysis

Bode plot diagrams were used to analyze the control loop design. When utilizing a Bode plot, we can determine the stability of the open loop transfer function by analyzing the phase margin and the gain margin. A Bode plot consists of two plots:

1) Bode magnitude plot

Logarithm of the magnitude or gain of sinusoidal transfer function

$$G_{(j\omega)} = 20\log_{10}|G(j\omega)|$$

2) Bode phase plot

Phase angle in degrees

$$\phi(\omega) = -\tan^{-1}\frac{1}{-\omega T}$$

Appendix B contains examples of Bode plots.

6.8.2 Phase and Gain Margin Analysis

In order to assess the stability of the open loop control function, it was required to determine the gain and phase margin. A system is stable when both gain and phase margin are positive. Negative margin values indicate an unstable system. When the steady state error and the transient response are in an uncertain state, the system is unstable. Alternatively, a stable system is a system that has a bounded response to a bounded input, and when time approaches infinity, the natural response approaches zero.

A stable landing system will guarantee a smaller overshoot. This is critical to avoid impact with the ground on descent. Oscillations are another side effect of an unstable system and can cause difficulties controlling the aircraft.

1) Gain margin (G_M)

   Gain margin can be determined by finding the phase angle of the frequency ωG_M is at value of 180^o. At this frequency, we calculate how much gain we need in order to raise the magnitude to 0 dB on the magnitude graph.

2) Phase margin (ϕ_M)

Phase margin can be determined by finding the frequency $\omega\phi_M$

on the magnitude graph when the gain is at 0 dB, as shown in Fig. 57. At this frequency, we calculate the difference between our phase value to 180^{o} on the phase graph.

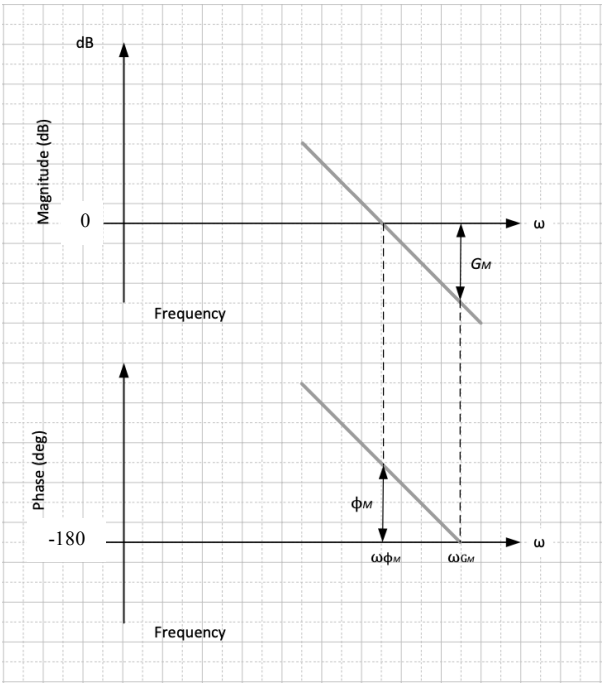


Fig. 57. Phase and gain margin.

Appendix C contains phase and gain margin examples.

6.8.3 Evaluation of the Transfer Function Via Step Response Input

Fig. 58 shows the MATLAB Simulink test setup for a step response input.

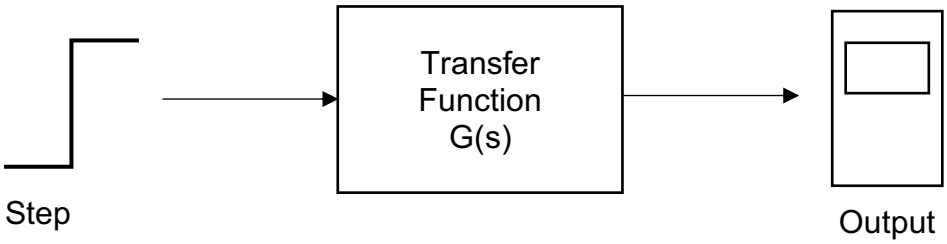


Fig. 58. Airplane transfer function step response test.

The step response in Fig. 59 shows a slow settling time.

This kind of response is not desirable in an aircraft control system as it is too

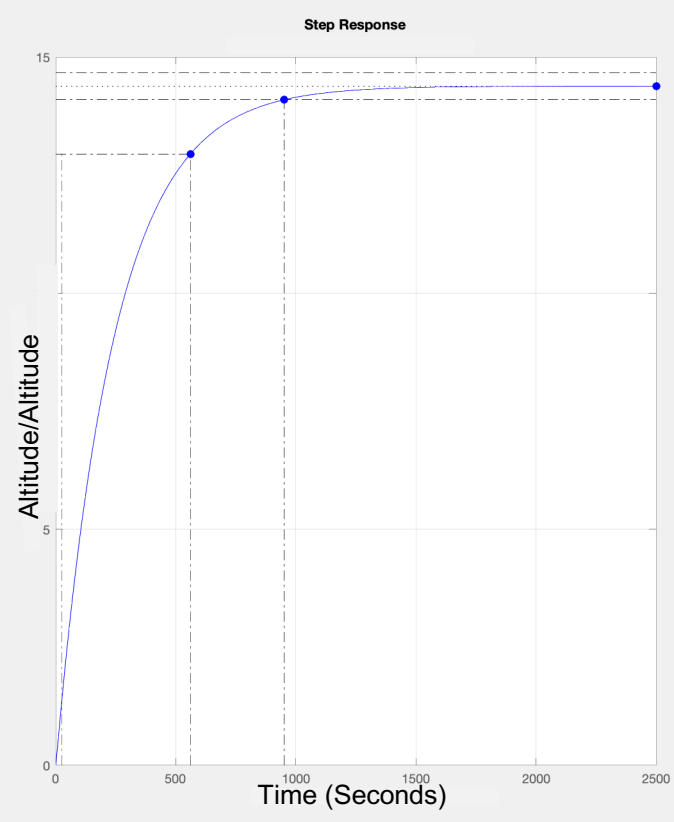


Fig. 59. Airplane transfer function output time step response.

slow to react and will not enable a successful tracking of a target landing trajectory.

6.8.4 Stability Analysis Using Bode Plots

The Bode plot in Fig. 60 has the characteristics of a low pass filter. The low pass filter shares the expected dynamics of an airplane as the gain portion (23 dB in this case) is a result of elevator input to altitude output. In low frequencies a DC gain is evident.

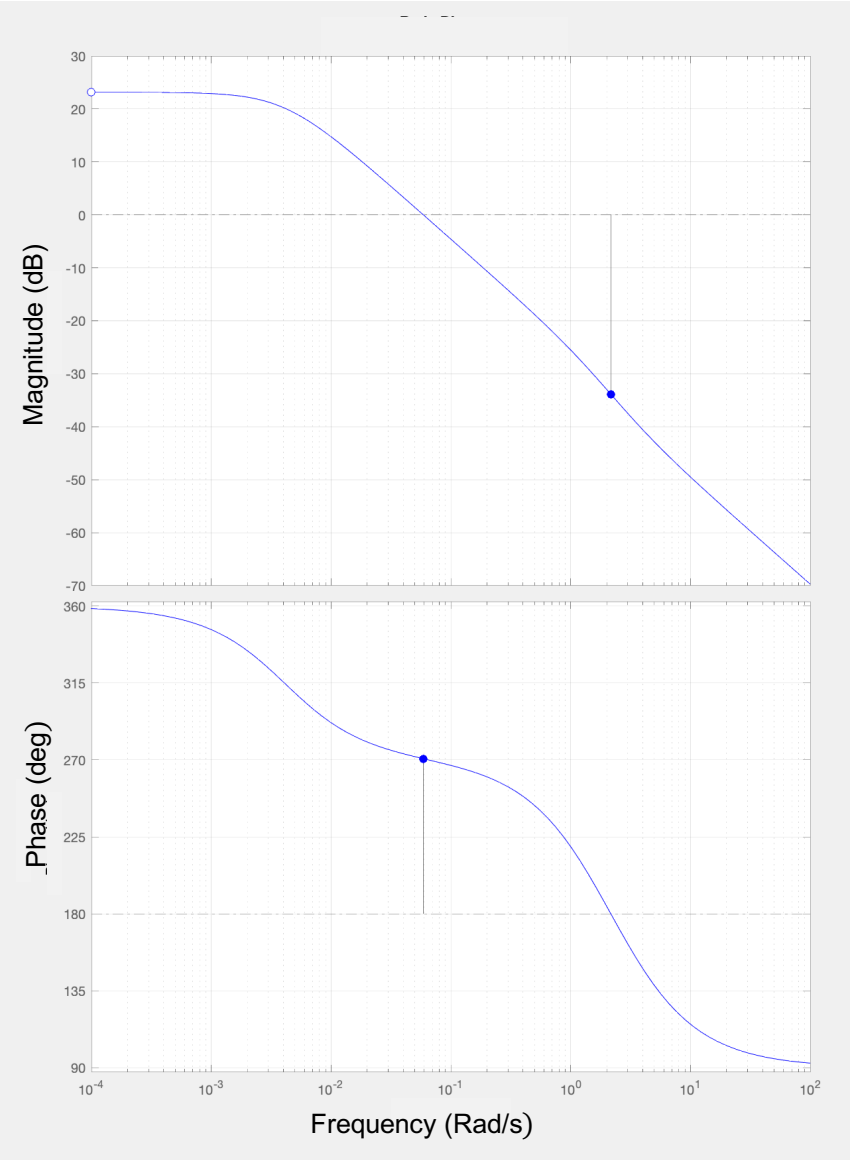


Fig. 60. Airplane transfer function bode plot.

This can be explained using the analogy of an amplifier with a ratio of input to output with input signal less than output. The elevator input behaves as the small input signal, and the altitude is the amplified signal. As the frequency increases, drop-in magnitude results as the aircraft becomes slower to react to the fast-changing inputs. This behavior is analogous to a low pass filter.

6.9 Steady State Error

6.9.1 Simulink Simulation of Transfer Function in a Control Loop

Analyzing the transfer function in a close loop using MATLAB Simulink in Fig. 61 and Fig. 62 shows the system has a steady state error.

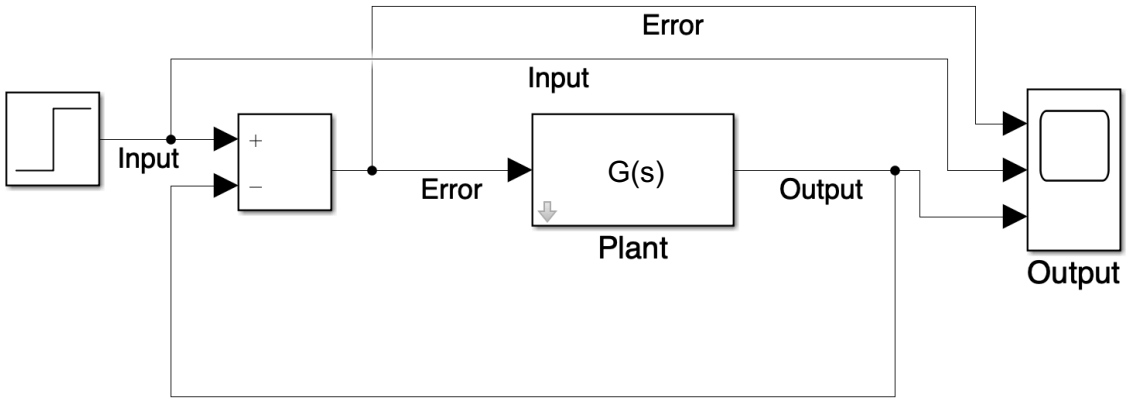


Fig. 61. Simulink diagram of the transfer function in loop.

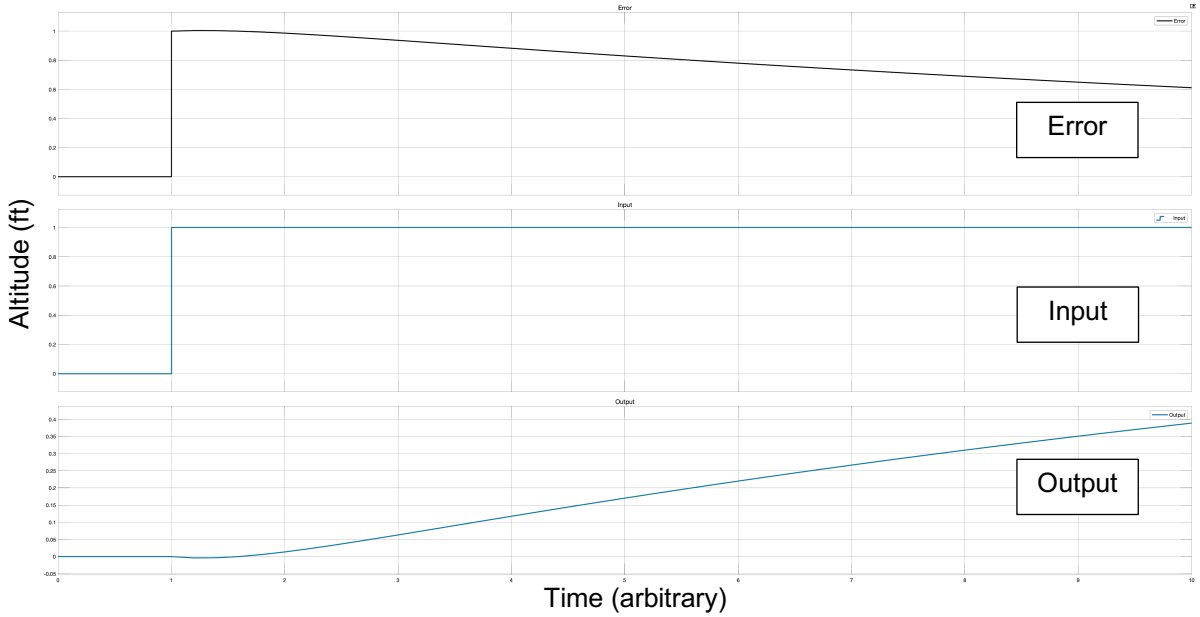


Fig. 62. Step response, output and error.

6.9.2 Steady State Error Analysis

Steady State error is the difference between the input and output for a prescribed test input.

as t -> ∞ [23].

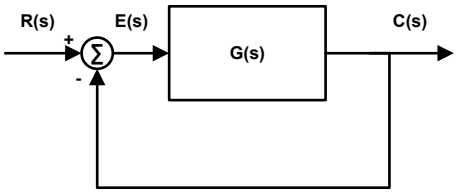


Fig. 63. Plant in a closed loop.

For the system shown in Fig. 63, the closed loop transfer function is:

$$\frac{C(s)}{R(s)} = \frac{G(s)}{1 + G(s)}$$

The transfer function for the error signal e(t) over the input signal r(t) is:

$$\frac{E(s)}{R(s)} = 1 - \frac{C(s)}{R(s)} = \frac{1}{1 + G(s)}$$

$$E(s) = \frac{1}{1 + G(s)} R(s)$$

Applying the final value theorem

$$e(\infty) = \lim_{t \to \infty} e(t) = \lim_{s \to 0} s\, E(s)$$

$$e(\infty) = \lim_{s \to 0} \frac{sR(s)}{1 + G(s)}$$

6.9.3 System Types

Control systems can be divided into types according to their ability to follow different kind of inputs, such as step, ramp, and parabolic. Consider the following open loop transfer function:

$$G(s) = \frac{K(T_a s + 1)(T_b s + 1) \dots (T_m s + 1)}{s^n (T_1 s + 1)(T_2 s + 1) \dots (T_p s + 1)}$$

The denominator term s^n represents a pole of multiplicity *N* at the origin. The system type is defined by the number of integrations of the open loop transfer function. A system is defined as type 0, type 1, type 2, etc., when *N* = 0, *N* = 1, *N* =2, respectively. The accuracy of the system, or the ability to follow an input, increases with the system type number, though increasing the system type potentially risks the system stability [21], as each higher type order adds an integrator. Each integrator adds a 90 degree shift in phase. The phase shift can cause instability when the phase margin becomes too small. Adding an additional zero for every type order solve the stability problem as it cancels the integrator.

6.9.4 Static Error Constants

Steady state error performance values are defined as *static error constants.* Assuming step input *u(t)* in time domain or $R(s) = \frac{1}{s}$ in the Laplace domain, then

$$e(\infty) = e_{step}(\infty) = \lim_{s \to 0} \frac{s(\frac{1}{s})}{1 + G(s)} = \frac{1}{1 + \lim_{s \to 0} G(s)}$$

Assuming ramp input *tu(t)* in time domain or $R(s) = \frac{1}{s^2}$ in the Laplace domain, then

$$e(\infty) = e_{ramp}(\infty) = \lim_{s\to 0} \frac{s(\frac{1}{s^2})}{1 + G(s)} = \lim_{s\to 0} \frac{1}{s + sG(s)} = \frac{1}{\lim_{s\to 0} sG(s)}$$

Assuming parabolic input $\frac{1}{2}t^2u(t)$ or $R(s) = \frac{1}{s^3}$ in the Laplace domain, then

$$e(\infty) = e_{parabolic}(\infty) = \lim_{s\to 0} \frac{s(\frac{1}{s^3})}{1 + G(s)} = \lim_{s\to 0} \frac{1}{s^2 + s^2G(s)} = \frac{1}{\lim_{s\to 0} s^2G(s)}$$

The static error constant is defined using the denominator terms as they are taken to the limit [23].

The static error constants for the three inputs of step, ramp, and the parabolic function are

Position constant $K_p = \lim_{s\to 0} G(s)$

Velocity constant $K_v = \lim_{s\to 0} sG(s)$

Acceleration constant $K_a = \lim_{s\to 0} s^2G(s)$

Table 4
Error and System Types [23]

input	Steady state constant	Type 0		Type 1		Type 2	
		Static error constant	error	Static error constant	error	Static error constant	error
step *u(t)* $R(s)=\frac{1}{s}$	$\frac{1}{1+K_p}$	$K_p = constant$	$\frac{1}{1+K_p}$	$K_p=\infty$	0	$K_p=\infty$	0
ramp *tu(t)* $R(s)=\frac{1}{s^2}$	$\frac{1}{K_v}$	$K_v=0$	∞	$K_v = constant$	$\frac{1}{K_v}$	$K_v=\infty$	0
parabolic $\frac{1}{2}r^2u(t)$ $R(s)=\frac{1}{s^3}$	$\frac{1}{K_a}$	$K_a=0$	∞	$K_a=0$	∞	$K_a = constant$	$\frac{1}{K_a}$

The term

$$\lim_{s\to 0} G(s)$$

is the DC gain of the forward transfer function, since S, the frequency variable, is approaching zero to confine the equation to zero stead-state error.

Hence, to satisfy the equation

$$\lim_{s\to 0} G(s)=\infty$$

$G(s)$ must take on the following form

$$G(s)=\frac{(s+z_1)(s+z_2)\ldots}{s^n(s+p_1)(s+p_2)\ldots}$$

in order for the limit to be infinite, the denominator must equal zero as S goes to zero.

Thus n ≥ 1; that is, at least one pole must be at the origin. Since division by S in the frequency domain is integration in the time domain. This means at least one pure integration must be present in the forward path.

Without an integrator when n = 0, we have

$$\lim_{s\to 0} G(s) = \frac{z_1 z_2 \ldots}{p_1 p_2 \ldots}$$

which results in a finite error.

The conclusion is that for a step input into unity feedback system, the steady state error will be zero only if there at least one pure integration in the forward path. If there is no integration, there will be a none-zero finite error [23].

6.10 Control System and Compensator Design

The following steps demonstrate the development process to define the compensator. The goals for the compensator design are zero steady state error, fast settling time and stability. The first step was to analyze the plant transfer function, as shown in Fig. 64 and discussed in Section “Evaluation of the Transfer Function Via Step Response Input.” The step response resulted in slow settling time and a steady state error. As discussed in the Section “Static Error Constants,” at least one pure integrator is required to be present in the forward

path to drive the steady state error to zero.

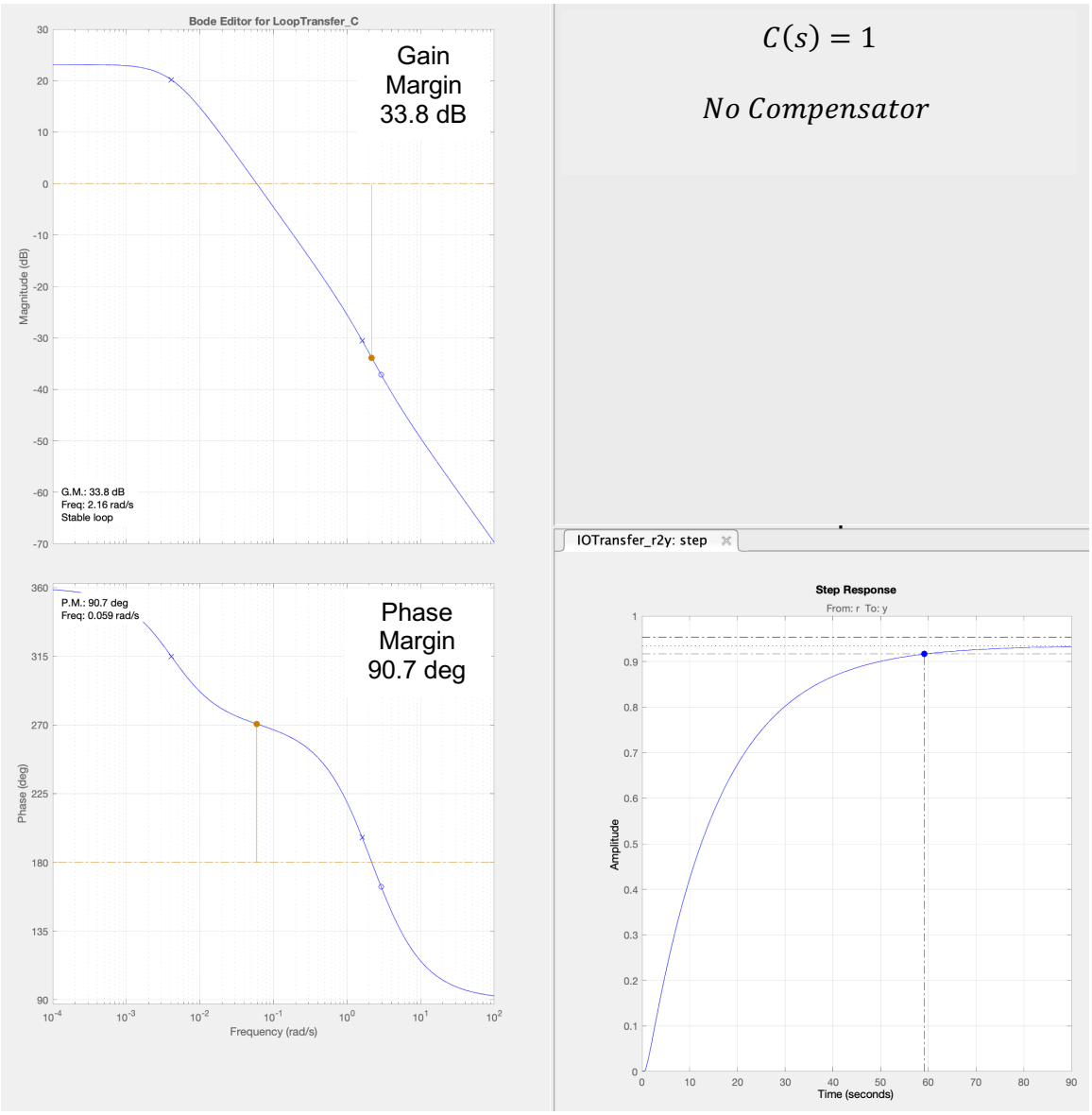


Fig. 64. Transfer function Bode plot without compensator.

With a pure integrator when $C = \frac{1}{S}$ as shown in Fig. 65, the system was unstable as the integrator added a 90 degree shift. The result is the gain and phase margins were negative, and the step response never settled.

Table 5
Compensator Dynamics with an Integrator

Type	Location	Damping	Frequency
Integrator	0	-1	0

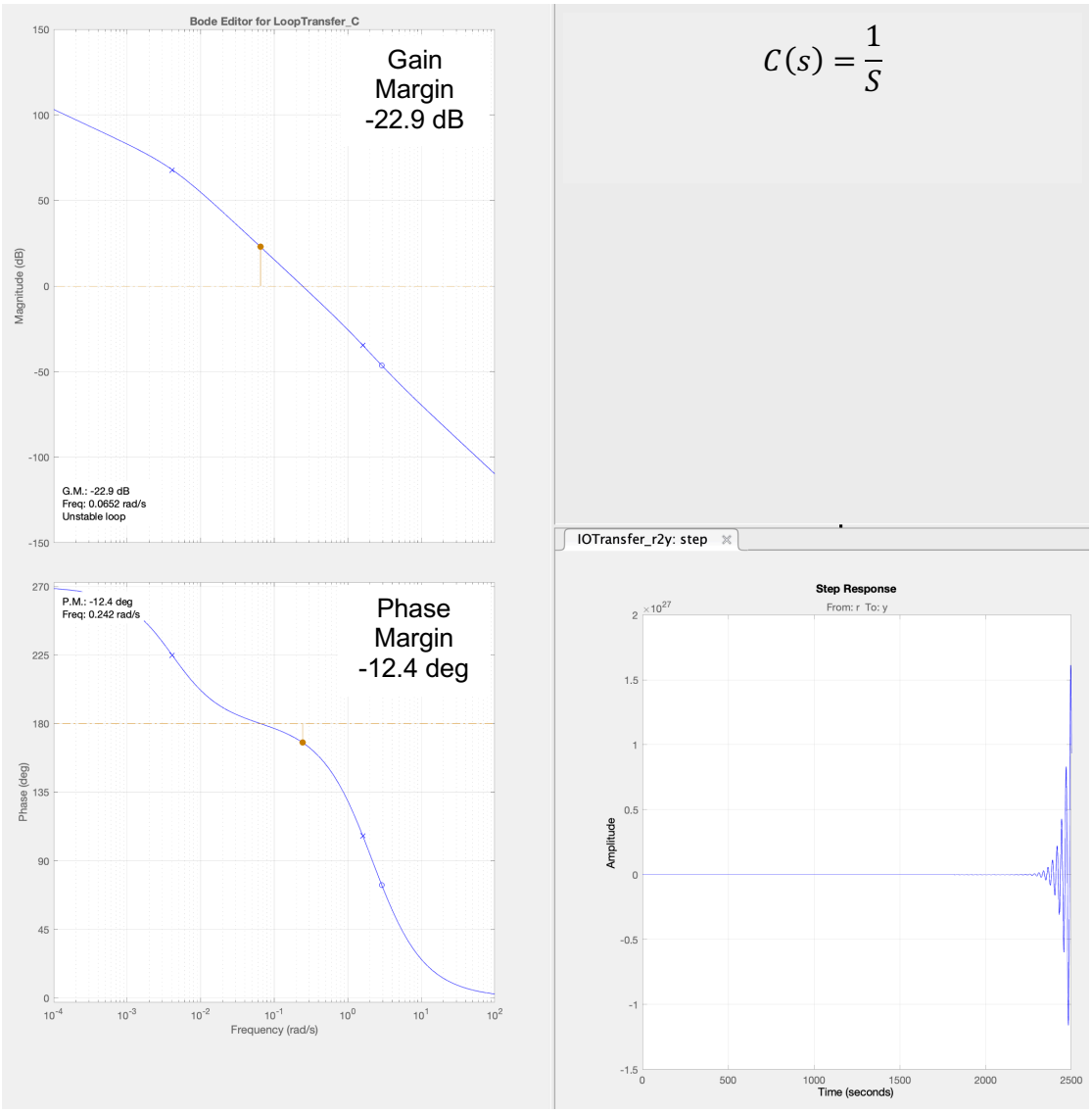


Fig. 65. Forward path Bode plot with integrator compensator.

The next step to improve stability was to reduce gain to affect an increase in gain and phase margin as shown in Fig. 66. As gain was reduced, however, bandwidth was also reduced. Lower bandwidth caused a significant increase in the step response settling time.

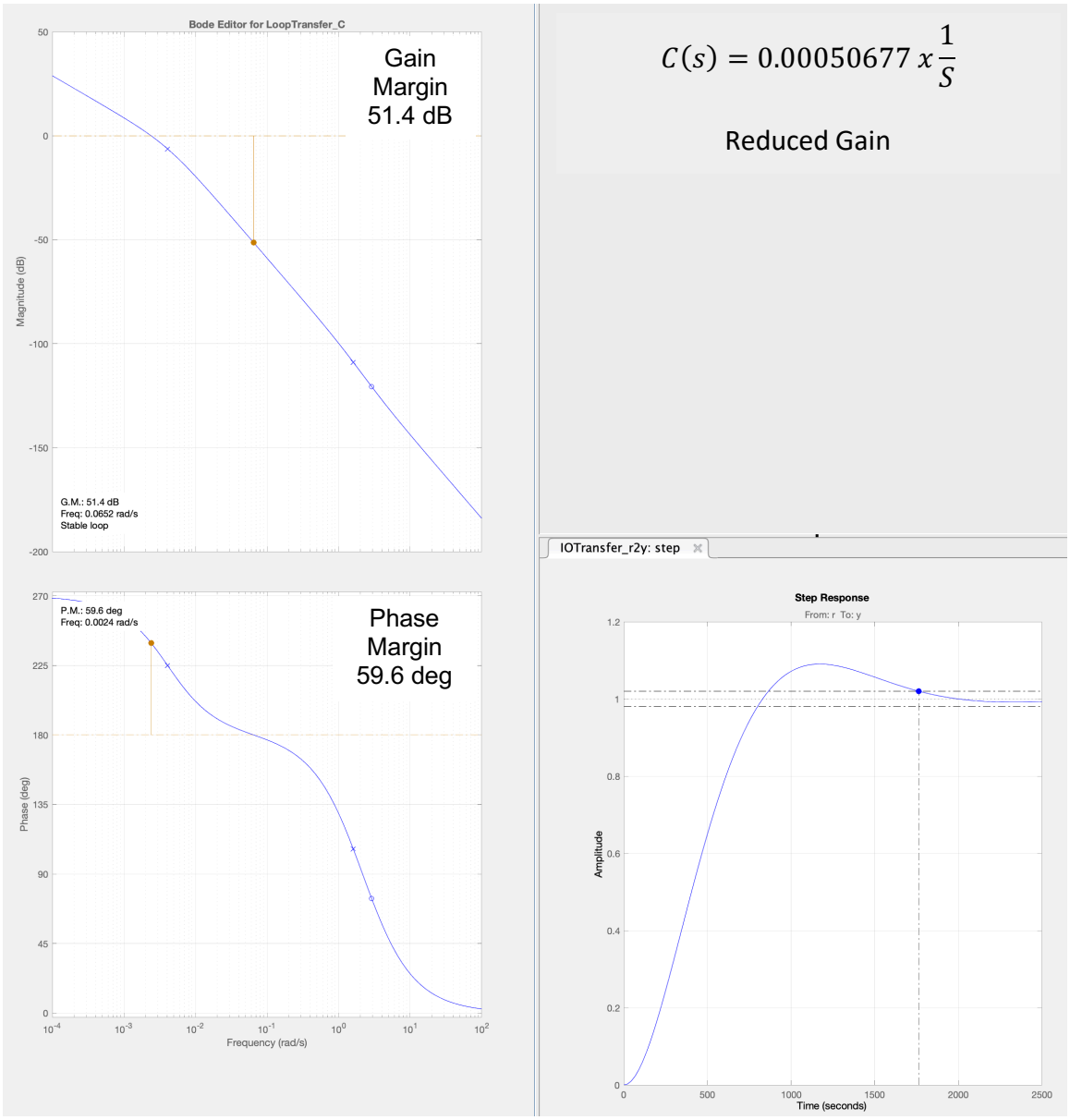


Fig. 66. Forward path Bode plot with reduced gain integrator compensator.

The next step was to add a zero, as the addition of a zero increases our phase margin by 90 degrees. As Fig. 67 details, the addition of a zero improved the gain margin.

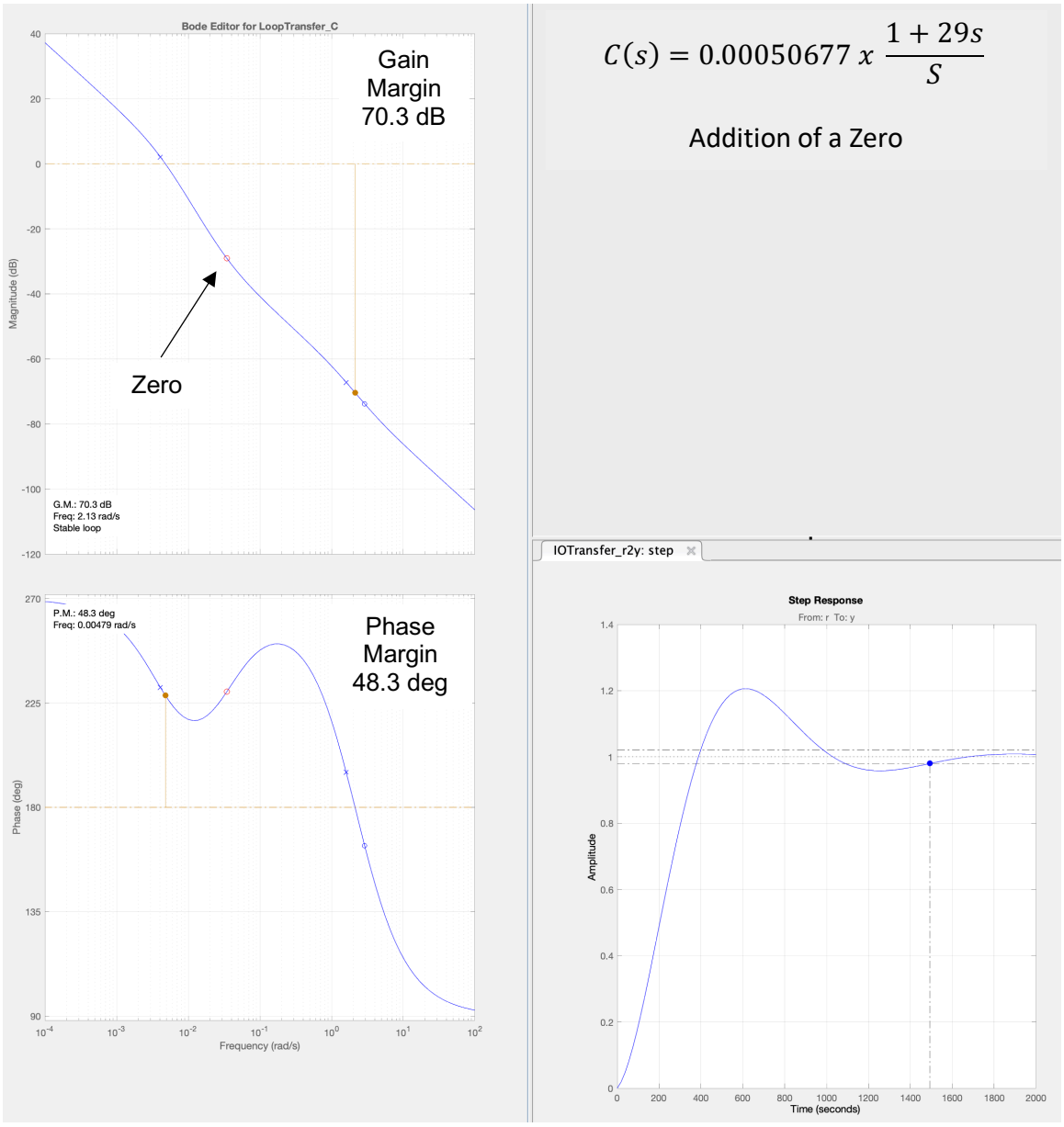


Fig. 67. Forward path Bode plot with integrator compensator and a zero.

Table 6
Compensator Dynamics with a Zero

Type	Location	Damping	Frequency
Integrator	0	-1	0
Real Zero	-0.0346	1	0.0346

Improved gain margin allows an increase in gain. As previously mentioned, an increase in gain decreases stability but with the benefit of faster settling time of the step response.

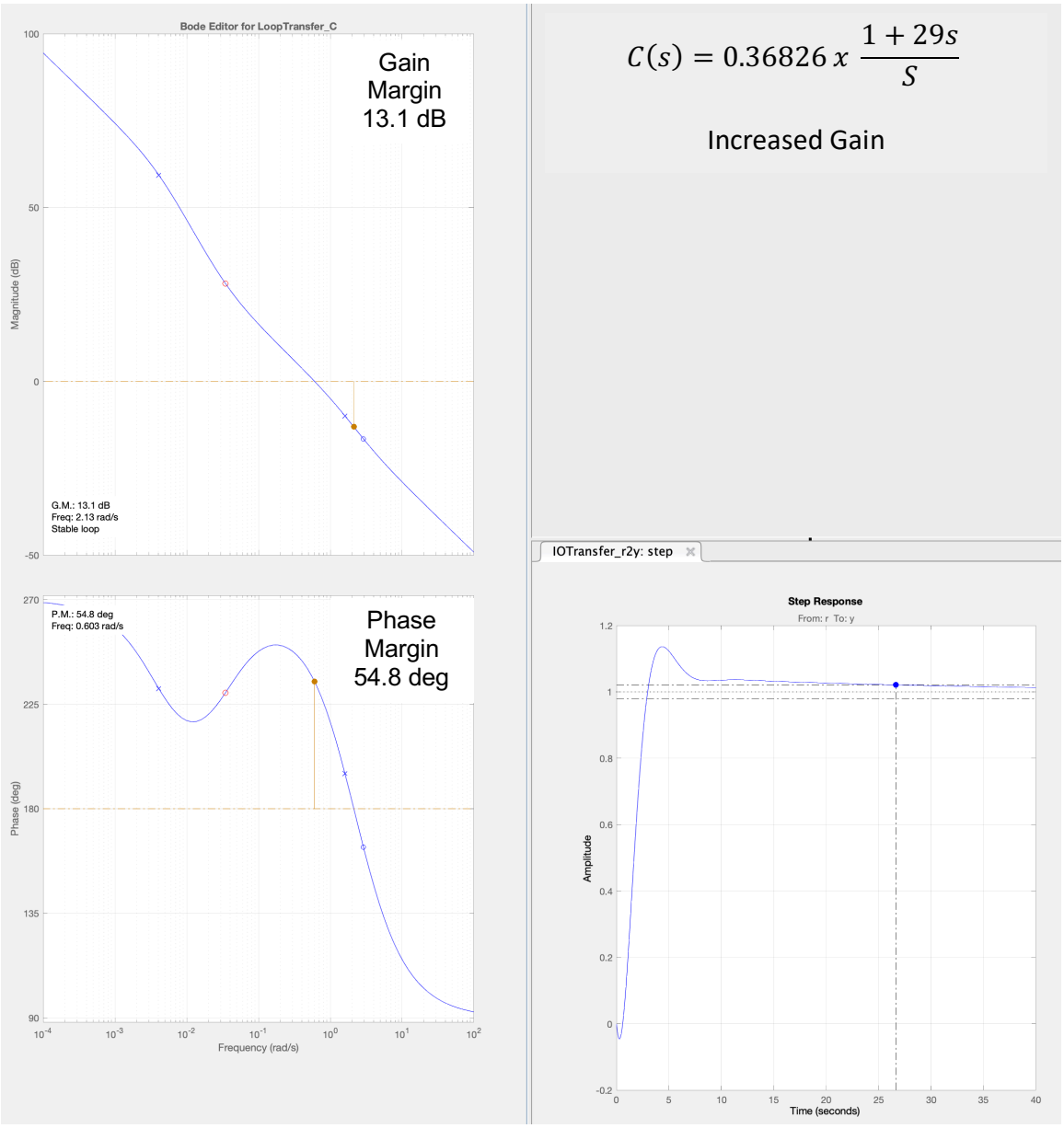


Fig. 68. Forward path Bode plot with integrator compensator and increased gain.

The last step was to add a pole to attenuate high frequency noise. With this method, the closed loop acted like low pass filter. The added pole decreased the settling time of the step response.

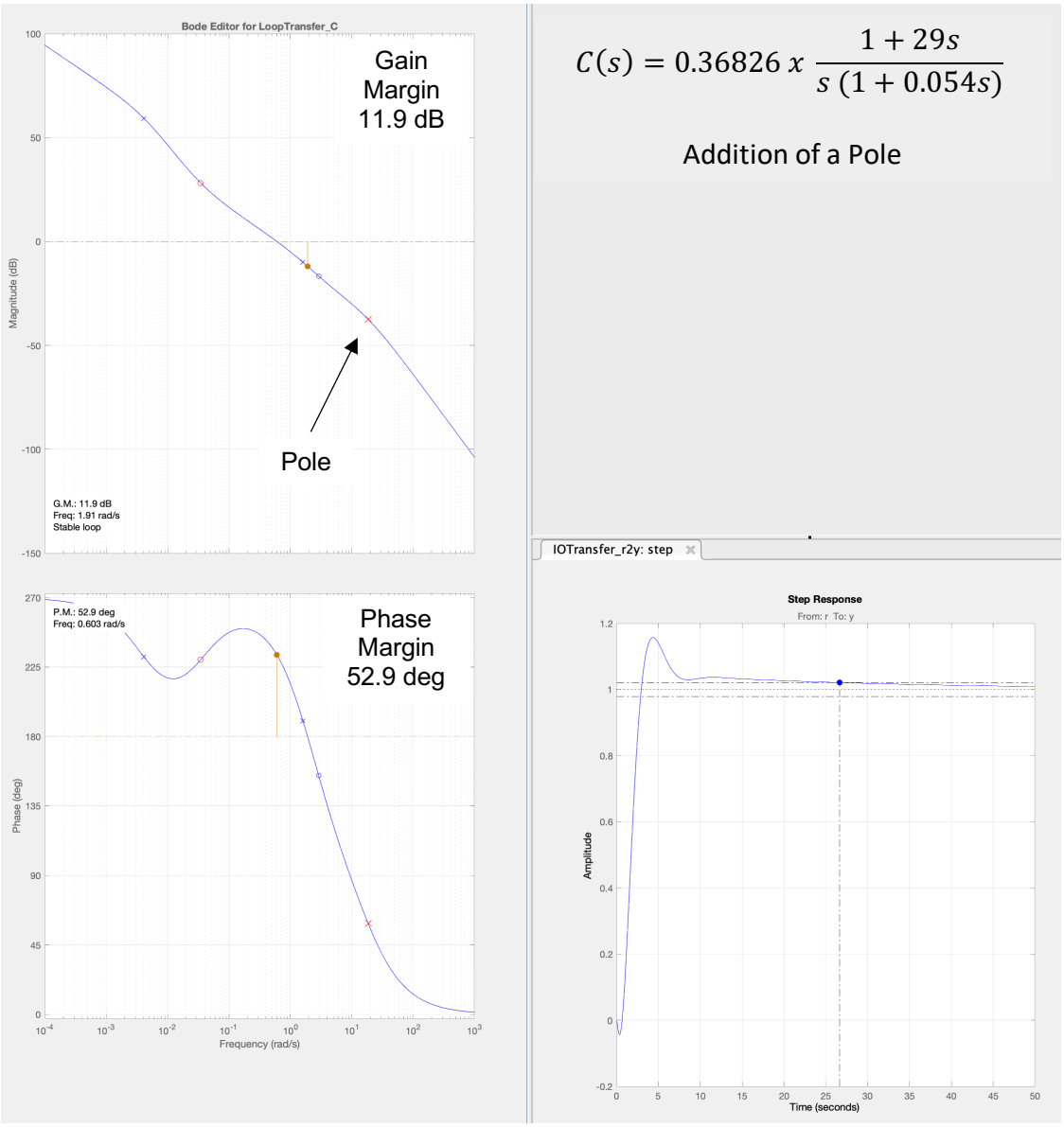


Fig. 69. Forward path Bode plot with the final compensator design.

Table 7
Compensator Dynamics with a Pole

Type	Location	Damping	Frequency
Integrator	0	-1	0
Real Zero	-0.0346	1	0.0346
Real Pole	-18.6	1	18.6

The compensator:

$$C = 0.36826\text{x}\frac{1}{S}\text{x}\frac{(1+29\,S)}{(1+0.054\,S)}$$

Fig. 70 details the Bode plot for the compensator integration aspect and steady state error. A zero was added to improve phase margin, and a high frequency pole was added to augment closed loop filtering.

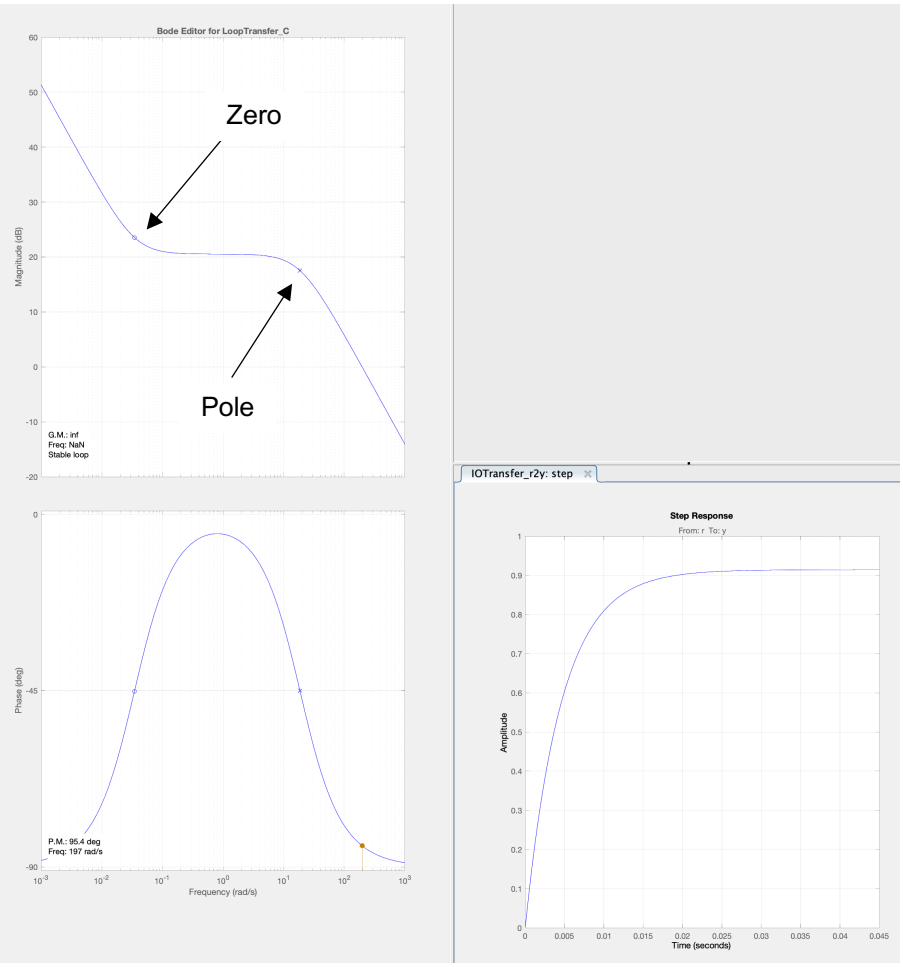


Fig. 70. Bode compensator plot.

Fig. 71 demonstrates that the closed loop with this compensator achieved a step response with 90% of desired amplitude within a settling time of 7 second.

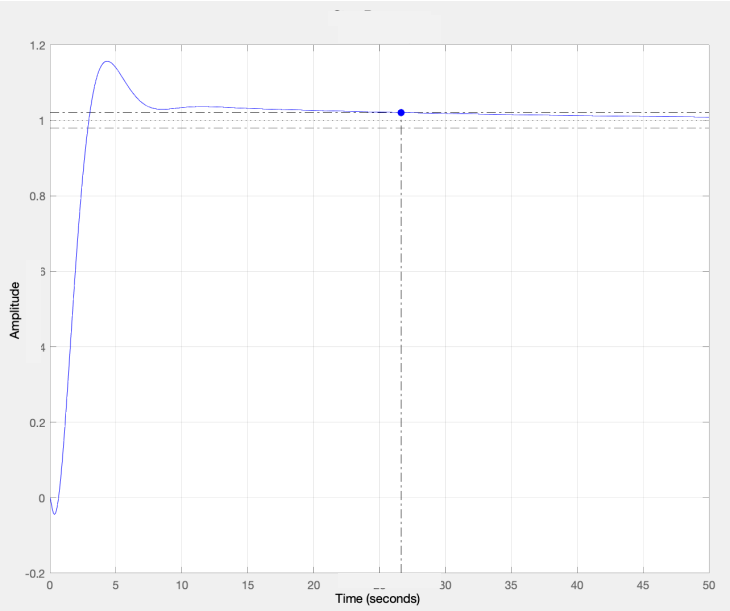


Fig. 71. Response of control loop with a compensator integrator.

6.10.1 Adding a Compensator to The Control Loop

Adding a compensator C(s) to the transfer function G(s):

$$Forward\ TF(s) = G(s)C(s) = \frac{-0.03266s + 0.09495}{s^2 + 1.609s + 0.006601} \text{x} \frac{198\ \text{s} + 6.862}{s^2 + 18.63\ s} =$$

$$= \frac{-6.467s^2 + 18.58s + 0.6515}{s^4 + 20.24s^3 + 29.97s^2 + 0.123s}$$

Closing the loop:

$$Closed\ Loop\ TF(s) = \frac{Forward\ TF(s)}{1 + Forward\ TF(s)} =$$

$$= \frac{\frac{-6.467s^2 + 18.58s + 0.6515}{s^4 + 20.24s^3 + 29.97s^2 + 0.123s}}{1 + \frac{-6.467s^2 + 18.58s + 0.6515}{s^4 + 20.24s^3 + 29.97s^2 + 0.123s}} =$$

$$= \frac{-6.467s^6 - 112.3s^5 + 182.8s^4 + 569.3s^3 + 21.81s^2 + 0.08013s}{s^8 + 40.48s^7 + 463.1s^6 + 1101s^5 + 1086s^4 + 576.7s^3 + 21.81s^2 + 0.08013s}$$

Fig. 72 exhibit the stability of the close loop system as gain margin is 12 dB and the phase margin is 63 degrees.

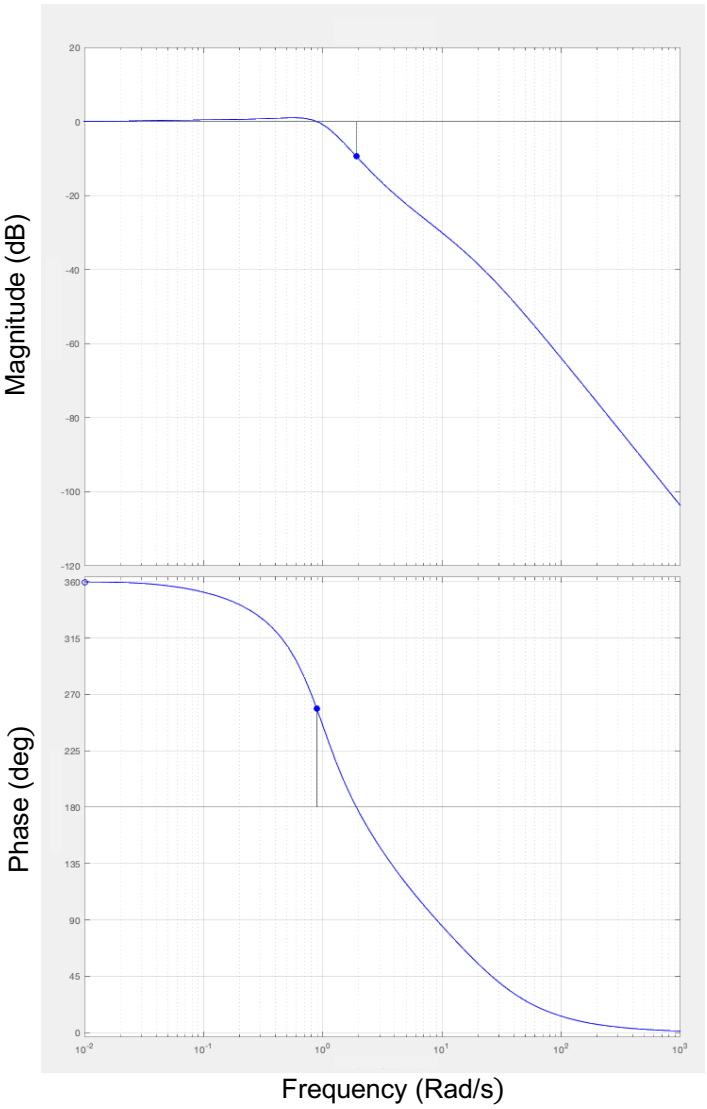


Fig. 72. Bode plot of the closed loop system.

6.10.2 Simulink Simulation of Transfer Function in a Control Loop with Compensator

Adding an integrator compensator removed the error from our close loop system. Adding a compensator to the system as shown in Fig. 73 reduced the error significantly.

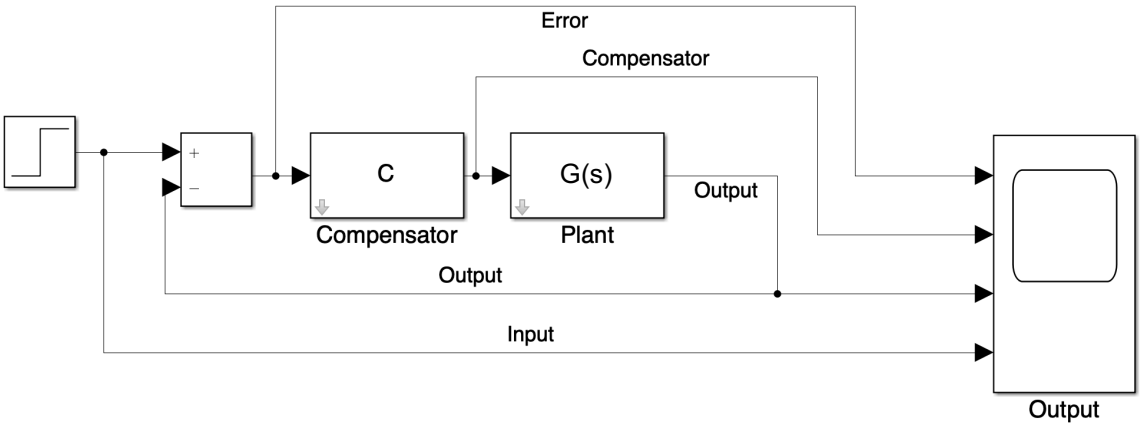


Fig. 73. Transfer function with a compensator in the loop.

The error graph below shows how the error value drops to an acceptable level quickly. The output graph in Fig. 74 shows how the output follows the step input.

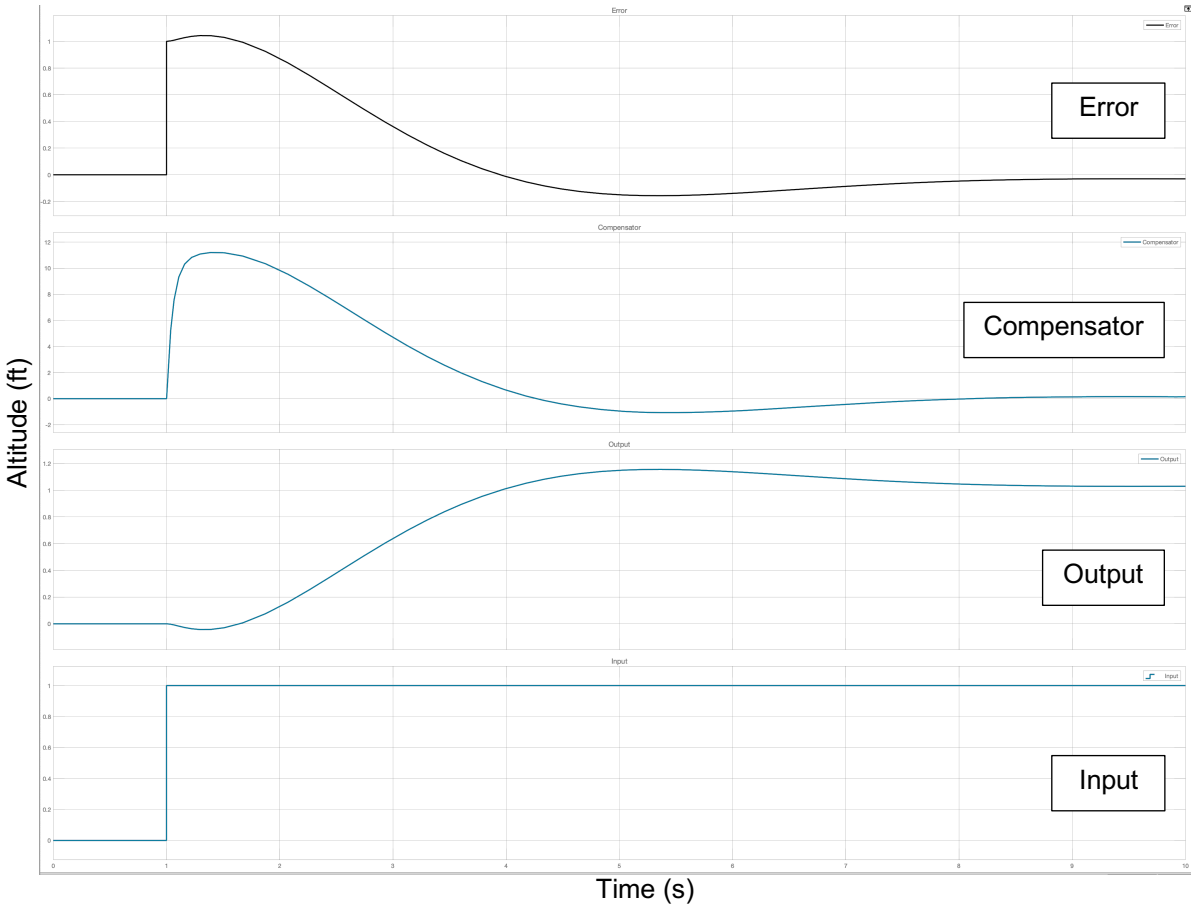


Fig. 74. Step response, output, compensator and error.

6.10.3 Landing Simulation of the Closed Loop Transfer Function

The landing trajectory graph in Fig. 75 and Fig. 76 shows that the error in the desired altitude is 0.677 feet or 8.124 inches.

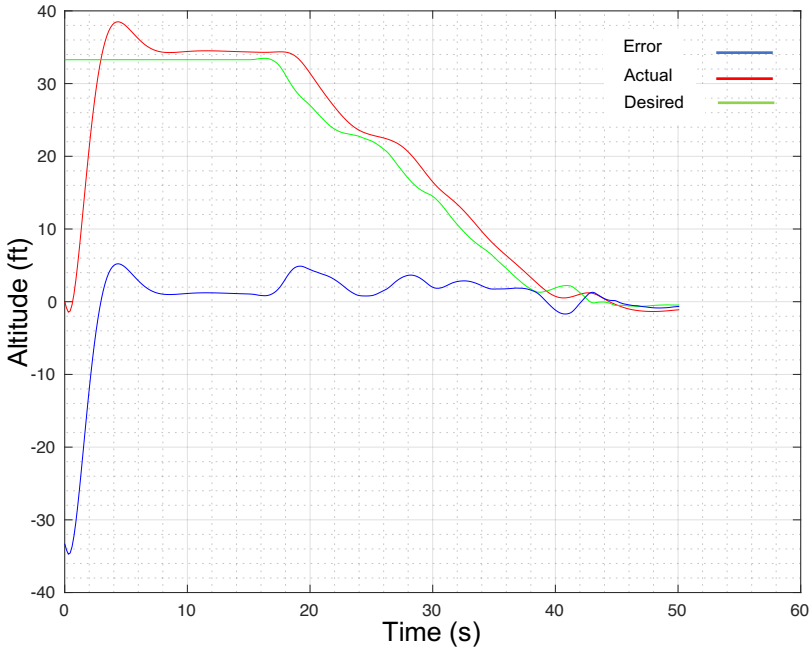


Fig. 75. Landing simulation with experimental landing data trajectory.

For the aircraft under test this is an acceptable error margin, as at this point of the landing the airplane is in a slow gliding, low slope decent without motor power. Additionally, at altitudes below 4 feet the auto-landing controller will initiate the flare maneuver using the LIDAR sensor.

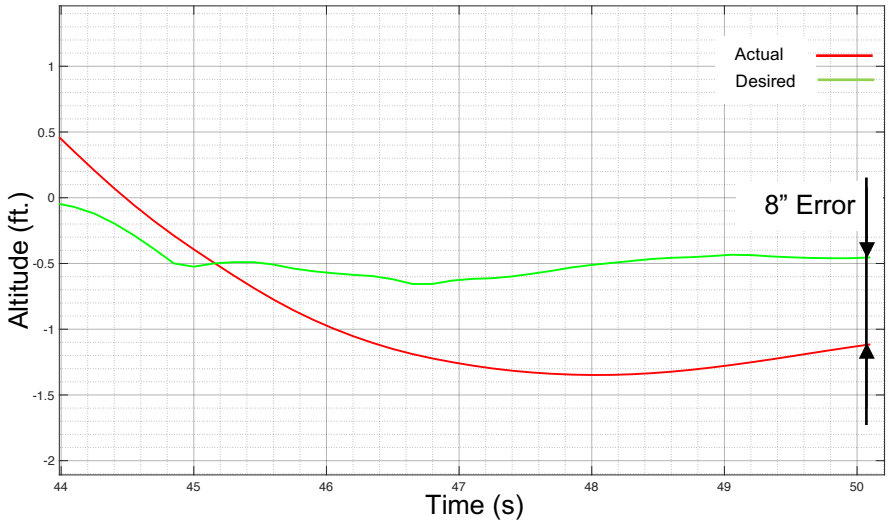


Fig. 76. Landing simulation zoomed in.

6.11 Laplace to Z Transform of the Compensator for Microcontroller Integration

6.11.1 Frequency conversion into Discrete Time Domain

Similarly to the Laplace conversion of continuous time to frequency, z transforms are used to convert discrete time domain into discrete frequency domain. Analog system stability and response are determined by the gain and component values. Although digital systems stability and system response are affected by gain and component values in the same way, they are each affected by the sampling and sampling rate. A desired transformation will be in a form of a transfer function that contains all the information of the sampled data. An ideal sampler, r(t), contains a series of impulses with values *r(KT),* we get

$$r(t) = \sum_{k=0}^{\infty} r(KT)\delta(1 - KT),$$

When t > 0, the Laplace transform of *r(t)*:

$$\mathcal{L}r(t) = \sum_{k=0}^{\infty} r(KT)e^{-KsT}$$

$\mathcal{L}r(t)$ is an infinite series contains multiples of e^{sT} and its powers.

Assuming a conformal mapping from S to Z

$$Z = e^{sT}$$

A Z transform will be

$$\mathcal{Z}\{r(t)\} = \sum_{k=0}^{\infty} r(KT)z^{-K}$$

Example of unit step function *u(t)* Z transform

$$\mathcal{Z}\{u(t)\} = \sum_{k=0}^{\infty} u(KT)z^{-K} = \sum_{k=0}^{\infty} z^{-K}$$

As *u(KT)* = 1 when $K \geq 0$. And in closed terms:

$$U(z) = \frac{1}{1 - z^{-1}} = \frac{1}{Z - 1}$$

6.11.2 Utilizing MATLAB Z Transform

MATLAB Toolbox c2d is a Z-transform tool capable of converting continuous time into discrete time. The tool utilizes the Zero-Order-Hold method to perform this transformation. For a staircase input, a Zero-Order-Hold (ZOH) method can convert between discrete and continuous time in the time domain. Fig. 77 demonstrates the concept of Zero-Order-Hold (ZOH) discretization $H_d(z)$ of a continuous-time model $H(s)$.

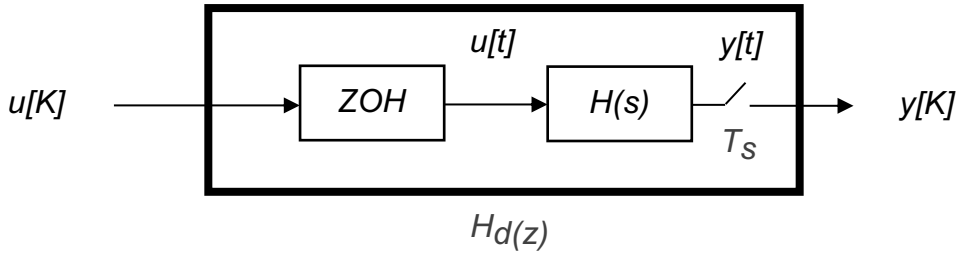


Fig. 77. MATLAB Zero-Order-Hold.

The ZOH module creates a continues-time $u(t)$ signal as it holds the sample input $u(k)$ constant over each sample period.

$$u(t)=u[k], \qquad kT_S \le t \le (k+1)T_S$$

u(t) is the input into $H(s)$ a continues system. $y[k]$ is a result of sampling $y(t)$ every T_s seconds.

Fig. 78 and Fig. 79 shows a system with internal delay. Such a system will result in an approximate discretization.

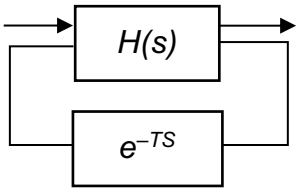


Fig. 78. MATLAB Zero-Order-Hold delay.

This tool will perform a few actions to approximate the zero-order-hold discretization to transform a system with delays.

The delay is decomposed $\tau=kT_S+\rho$ with $0 \le \rho < T_S$.

The delay ρ *is absorbed into* $H(s)$, discretizing $H(s)$ to $H(z)$.

The integer portion of kT_s is represented in form of internal discrete time delay z^{-k} [22]

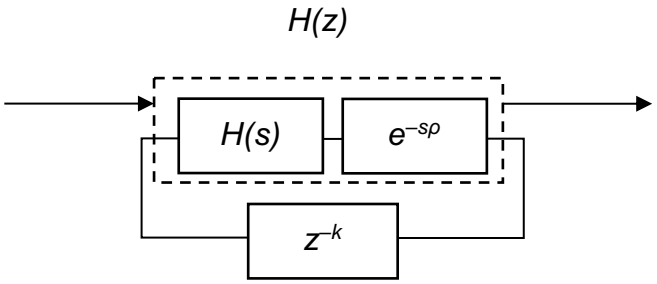


Fig. 79. MATLAB Zero-Order-Hold discrete delay.

6.11.3 S to Z Transformation of the Compensator

Applying the MATLAB c2d tool on the continuous-time closed loop transfer function:

$$Compensator\ C(s) = 0.36826\text{x}\frac{1}{S}\text{x}\frac{(1+29S)}{(1+0.054S)}$$

resulted in the following discrete time Z transform compensator:

$$Compensator\ C(z) = \frac{9.007\ z - 8.976}{z^2 - 1.156z + 0.1557}$$

Sample time: 0.1 seconds

Discrete-time transfer function.

7 SUMMARY

The results of this thesis demonstrate successful execution of the unmanned aerial vehicle landing-control algorithm. Experimental flight trajectory data were used in simulations, and the control system tracked the desired trajectory successfully. The functionality of all hardware componentry was demonstrated experimentally. The project scope was successfully defined and managed by approaching the problem incrementally, fixing the airframe design, and using off-the-shelf componentry without the use of ground equipment. Even with the limitations of hardware speed and reaction time that were a consequence of this approach, simulations showed a small steady state error within acceptable limits. This system as defined and designed should be capable of performing an auto-landing in a repeatable and reliable fashion.

8 LIMITATIONS

The developed system was built around a specific commercially available airframe with additional sensors as described herein. To utilize the unmanned aerial vehicle landing-control algorithm on a different fixed-wing platform, experimental flight testing and data analysis would be required to define a unique airplane transfer function and matching control system. Furthermore, because the landing waypoint is preloaded into the auto-pilot before flight, the unmanned aerial vehicle landing-control algorithm is specific to that desired landing location. The landing location used in this application also assumes sufficiently long, obstruction-free runway surface. Inclement weather conditions such as high winds, rain, and snow were considered beyond the scope of this project.

9 CONCLUSIONS

Technological advancements in recent years, for example miniaturized powerful microcontrollers, flight controllers, sophisticated sensors, and the Global Positioning System, have enabled the development of platforms that can sense, calculate, and react. Additionally, software development tools for microcontrollers, control systems and flight simulation software are widely available. These many recent developments make it possible to design a fixed-wing platform that can be operated autonomously. This thesis focused on the development of a UAV landing algorithm for use with a simple cost-effective platform that, with some added componentry and modifications, is capable of flying autonomously, locating the runway, and landing safely. The development of affordable and easy to pilot unmanned autonomous vehicles can be enabled by technologies as proven in this thesis. The availability of powerful development and analysis tools such as MATLAB and flight simulators contribute to accessibility.

9.1 Solution and Conclusions

A few challenges had to be resolved during the development of this auto landing system. The transfer function of the fixed-wing airframe is not typically provided by the manufacturer, so it was required to be resolved experimentally as detailed in “Transfer Function Research and Development.”

The slow refresh rate of the Global Positioning System sensor made deriving a solution with reasonable settling time difficult; this was resolved by fine tuning the

control loop as covered in “Control System and Compensator Design.” Telemetry data logs were not time synchronized, requiring manipulation of the telemetry data as covered in “MATLAB Script and Data Interpolation.” The serial telemetry bus via MavLink did not operate as expected, causing issues with the auto-landing CPU interpreting telemetry. Debugging the load on the auto-landing CPU resolved the problem, with more information covered in “MavLink Connectivity Testing.”

9.2 Future Work and Improvements

The algorithm in this thesis concentrates on landing a fixed-wing airplane utilizing two microcontrollers. The auto-landing controller decides when to land, takes control of the motor and elevator, and initiates a pre-determined landing procedure. The auto-pilot controller is responsible for flying the plane to waypoints and for maintaining the stability of the airplane. Future work can expand upon the level of control assumed from the auto-pilot to the auto-landing controller. In the same fashion that the elevator and throttle were in this thesis, the aileron and rudder functions can be added to a control algorithm. Additional improvements to the control algorithm can be explored with respect to less than ideal environmental conditions, such as gusty tailwinds, crosswinds, or precipitation. A more robust management of typical environmental conditions would yield a control system that can remain stable while handling the larger perturbations caused by wind or equipment malfunctions. Additional improvements are likely if performing the polls and zeros simplification on the

plant transfer function before closing the loop. This would result in a plant transfer function of lower order and yield smaller steady state error. Additional information on pole and zero optimization can be found in Appendix D.

Literature Cited

[1] D. Zhang and X. Wang, “Autonomous Landing Control of Fixed-wing UAVs: from Theory to Field Experiment,” *Journal of Intelligent & Robotic Systems*, vol. 88, no. 2-4, pp. 619–634, 2017.

[2] A. K. Tripathi and R. Padhi, “Autonomous Landing for UAVs using T-MPSP Guidance and Dynamic Inversion Autopilot,” *IFAC-PapersOnLine*, vol. 49, no. 1, pp. 18–23, 2016.

[3] B. Cheng and Z. Guo, "Study on Small UAVs' Deep Stall Landing Procedure," 2017 5th International Conference on Mechanical, Automotive and Materials Engineering (CMAME), Guangzhou, 2017, pp. 269-274.

[4] A. Brezoescu, T. Espinoza, P. Castillo, and R. Lozano, “Adaptive Trajectory Following for a Fixed-Wing UAV in Presence of Crosswind,” *Journal of Intelligent & Robotic Systems*, vol. 69, no. 1-4, pp. 257–271, 2012.

[5] Q. Zhan, J. Wang and X. Xi, "Control system design and experiments of a quadrotor," 2012 IEEE International Conference on Robotics and Biomimetics (ROBIO), Guangzhou, 2012, pp. 1152-1157.

[6] J. Jiang, J. Qi, D. Song and J. Han, "Control platform design and experiment of a quadrotor," Proceedings of the 32nd Chinese Control Conference, Xi'an, 2013, pp. 2974-2979.

[7] F. Şenkul and E. Altuğ, "Modeling and control of a novel tilt — Roll rotor quadrotor UAV," 2013 International Conference on Unmanned Aircraft Systems (ICUAS), Atlanta, GA, 2013, pp. 1071-1076.

[8] Y. Choi and H. Ahn, "Formation control of quad-rotors in three dimension based on euclidean distance dynamics matrix," 2011 11th International Conference on Control, Automation and Systems, Gyeonggi-do, 2011, pp. 1168-1173.

[9] S. H. Jeong and S. Jung, "Design and control of a small quad-rotor system under practical limitations," 2011 11th International Conference on Control, Automation and Systems, Gyeonggi-do, 2011, pp. 1163-1167.

[10] C. Y. Chang-Sun Yoo and I. A. Iee-Ki Ahn, "Low cost GPS/INS sensor fusion system for UAV navigation," Digital Avionics Systems Conference, 2003. DASC '03. The 22nd, Indianapolis, IN, USA, 2003, pp. 8.A.1-8.1-9 vol.2.

[11] H. Lim, H. Lee and H. J. Kim, "Onboard flight control of a micro quadrotor using single strapdown optical flow sensor," 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura, 2012, pp. 495-500.

[12] A. Chan, S. Tan and C. Kwek, "Sensor data fusion for attitude stabilization in a low cost Quadrotor system," 2011 IEEE 15th International Symposium on Consumer Electronics (ISCE), Singapore, 2011, pp. 34-39.

[13] L. Kis and B. Lantos, "Quadrotor control based on partial sensor data," 19th International Workshop on Robotics in Alpe-Adria-Danube Region (RAAD 2010), Budapest, 2010, pp. 43-48.

[14] Y. K. Kwag and J. W. Kang, "Obstacle awareness and collision avoidance radar sensor system for low-altitude flying smart UAV," The 23rd Digital Avionics Systems Conference (IEEE Cat. No.04CH37576), Salt Lake City, UT, USA, 2004, pp. 12.D.2-121.

[15] Paul Infant Teenu Mohan Das, S. Swami and J. M. Conrad, "An algorithm for landing a Quadrotor Unmanned Aerial Vehicle on an oscillating surface," 2012 Proceedings of IEEE Southeastcon, Orlando, FL, 2012, pp. 1-4.

[16] J. Campbell, J. Hamilton, M. Iskandarani and S. Givigi, "A systems approach for the development of a quadrotor aircraft," 2012 IEEE International Systems Conference SysCon 2012, Vancouver, BC, 2012, pp. 1-7.

[17] F. Kendoul and K. Nonami, "A visual navigation system for autonomous flight of micro air vehicles," 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, St. Louis, MO, 2009, pp. 3888-3893.

[18] Wenhui Xiang, Yang Cao and Zengfu Wang, "Automatic take-off and landing of a quad-rotor flying robot," 2012 24th Chinese Control and Decision Conference (CCDC), Taiyuan, 2012, pp. 1251-1255.

[19] B. Herisse, F. Russotto, T. Hamel and R. Mahony, "Hovering flight and vertical landing control of a VTOL Unmanned Aerial Vehicle using optical flow," 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, 2008, pp. 801-806.

[20] “MAVLink Micro Air Vehicle Communication Protocol,” *QGround Control*. Accessed: May-2015. [Online]. Available: http://qgroundcontrol.org/mavlink/start.

[21] K. Ogata, *Modern Control Engineering*, Upper Saddle River, NJ, USA: Prentice Hall PTR, 2010.

[22] MATLAB. 9.7.0.1296695 (R2019b), The MathWorks Inc. Accessed: May-2020. [Online]. Available: https://www.mathworks.com/help.

[23] N. S. Nise, *Control systems engineering*, 6th ed. New York, NY: John Wiley & Sons, 2011.

[24] R. C. Dorf and R. H. Bishop, *Modern control systems*, 11th ed. Upper Saddle River, NJ: Pearson Education, 2008.

[25] "E-flite Apprentice® S 15e BNF Trainer Airplane with SAFE® Technology: Horizon Hobby," *E-flite Apprentice® S 15e BNF Trainer Airplane with SAFE® Technology | Horizon Hobby*. Accessed: May-2020. [Online]. Available: https://www.horizonhobby.com/apprentice-s-15e-bnf-with-safe-reg;-technology-efl3180?gclid=Cj0KCQjwj7v0BRDOARIsAGh37irlF8VaSS6tEbIf2ysLhng2TqxznHf6fgoUGNIGIU5skXiimCW7EL4aAkYiEALw_wcB.

[26] "Arm Mbed LPC1768 Board," *NXP*. Accessed: May-2020. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpc1700-cortex-m3/arm-mbed-lpc1768-board:OM11043.

[27] "mbed LPC1768 Schematic," *Mbed*. Accessed: May-2020. [Online]. Available: https://os.mbed.com/platforms/mbed-LPC1768/#schematics-and-data-sheets.

[28] "Pixhawk Flight Controller, 3DR," *3DR Pixhawk 1 · PX4 v1.9.0 User Guide*. Accessed: May-2020. [Online]. Available: https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk.html.

[29] "LEA-6 u-Blox 6 GPS Modules," *u-Blox*, Accessed: May-2020. [Online]. Available: www.u-blox.com/sites/default/files/products/documents/LEA-6_DataSheet_(UBX-14044797).pdf.

[30] "Garmin LIDAR-Lite v3: GPS Sensors," *Garmin*. Accessed: May-2020. [Online]. Available: https://buy.garmin.com/en-US/US/p/557294.

[31] "Airspeed Sensors," *Airspeed Sensors · PX4 v1.9.0 User Guide*. Accessed: May-2020.
[Online]. Available: https://docs.px4.io/v1.9.0/en/sensor/airspeed.html.

[32] "PPM Encoder," *3DR*. Accessed: May-2020.
[Online]. Available: https://www.onedrone.com/store/3dr-ppm-encoder-new.html.

[33] “Pololu 4-Channel RC Servo Multiplexer,” *Pololu Robotics & Electronics*. Accessed: May-2020. [Online]. Available: https://www.pololu.com/product/2806.

[34] “30-Amp Pro Switch-Mode BEC Brushless ESC,” *HorizonHobby*. Accessed: May-2020. [Online]. Available: https://www.horizonhobby.com/product/airplanes/airplane-accessories/electronic-speed-controls/30-amp-pro-switch-mode-bec-brushless-esc-efla1030?gclid=Cj0KCQjwj7v0BRDOARIsAGh37iqziAyUsKECkvVxB2OvibzX_Dexe-YU4GArfeL6iJfdKwa8c9XzKFQaAtCuEALw_wcB.

[35] “Brushless Outrunner Motor, 840Kv,” *HorizonHobby*. Accessed: May-2020. [Online]. Available: https://www.horizonhobby.com/brushless-outrunner-motor--840kv-eflm7215.

[36] “E-flite 3200mAh 3S 11.1V 20C LiPo RC Airplane or Heli Battery with EC3 Connector: Horizon Hobby,” *E-flite 3200mAh 3S 11.1V 20C LiPo RC Airplane or Heli Battery with EC3 Connector | Horizon Hobby*. Accessed: May-2020. [Online]. Available: https://www.horizonhobby.com/3200mah-3s-111v-20c-lipo--13awg-ec3-eflb32003s.

[37] “37g Standard Servo,” *HorizonHobby*. Accessed: May-2020. [Online]. Available: https://www.horizonhobby.com/37g-standard-servo-eflr7150.

[38] “13g Digital Micro Servo,” *HorizonHobby*. Accessed: May-2020. [Online]. Available: https://www.horizonhobby.com/13g-digital-micro-servo-eflr7155.

[39] “SUPER SLIM FM RECEIVER,” *Hitec RCD*, Accessed: May-2020. [Online]. Available: https://hitecrcd.com/files/ManualSuperslim.pdf.

Appendix A

A.1 Airframe [25]

Wingspan	59.0 in. (1500 mm)
Overall Length	42.5 in. (1080 mm)
Wing Area	515 sq. in. (33.2 sq. dm.)
Flying Weight	49.0 oz. (1390 g)
Motor Size	15-size brushless outrunner
CG (center of gravity)	3 1/8 in. of wing
Prop Size	11 x 8 in.

A.2 Auto-Landing Controller CPU [26]

ARM Cortex-M3 core	• 100 MHz operation • Nested vectored interrupt controller for fast deterministic interrupts • Wakeup interrupt controller allows automatic wake from any priority interrupt • Memory protection unit • Four reduced power modes: sleep, deep sleep, power-down, and deep power-down
Memories	• 512 kB of Flash memory • 64 kB of SRAM
Serial peripherals	• 10/100 Ethernet MAC • USB 2.0 full-speed device/Host/ OTG controller with on-chip PHY • Four UARTs with fractional baud rate generation, RS-48, modem control, and irda • Two CAN 2.0B controllers • Three SSP/SPI controllers • Three I^2C-bus interfaces with one supporting fast mode plus (1-Mbit/s data rates) • I^2S interface for digital audio
Analog peripherals	• 12-bit ADC with eight channels • 10-bit DAC
Other peripherals	Ultra low power (< 1 μA) RTC • General purpose DMA controller with eight channels • Up to 70 GPIO • Motor control PWM and quadrature encoder interface to support three phase motors • Four 32-bit general purpose timers/counters
Package	• 100-pin LQFP (14 x 14 x 1.4 mm)

A.3 Auto-Landing Controller Detailed Pin Assignments [27]

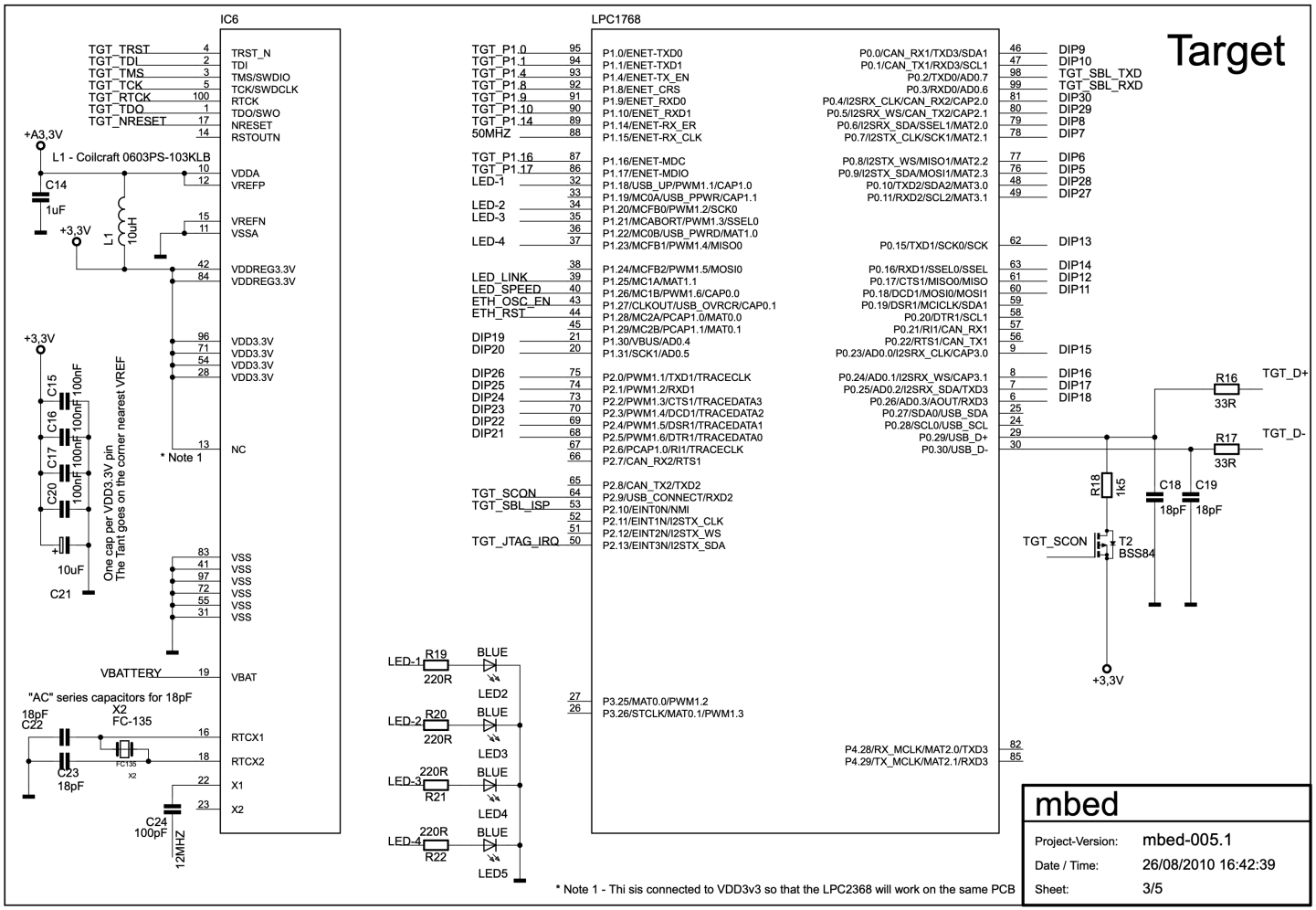


Fig. A1. Auto-landing controller pin assignment diagram.

A.4 Auto-Pilot [28]

Processor	
Model	32-bit STM32F427
Family	Cortex M4 core with FPU
Speed	168 MHz
Memory	256 kB RAM
Cash	2 MB Flash
Failsafe co-processor	32-bit STM32F103
Sensors	
Gyroscope	ST Micro L3GD20H 16-bit
Accelerometer/magnetometer	ST Micro LSM303D 14-bit
Barometer	MEAS MS5611
Interfaces	• 5x UART (serial ports), one high-power capable, 2x with HW flow control • 2x CAN (one with internal 4.3V transceiver, one on expansion connector) • Spektrum DSM / DSM2 / DSM-X® Satellite compatible input • Futaba S.BUS® compatible input and output • PPM sum signal input • RSSI (PWM or voltage) input • I2C • SPI • 3.3 and 6.6 V ADC inputs • Internal micro USB port and external micro USB port extension

A.5 GPS [29]

Model	ublox LEA-6H module
Speed	5 Hz update rate
Ceramic patch antenna size	25 x 25 x 4 mm
Filter	LNA and SAW f
Power	3 V lithium backup battery
Current	67 mA max
Regulator	Low noise 3.3 V
Interface	I2C
Total size	38 x 38 x 8.5 mm
Weight	0.59 oz. (16.8 g)

A.6 Proximity Sensor [30]

Range	0-40 m Laser Emitter
Accuracy	±0.025 m at distances greater than 1 m
Power	4.7 - 5.5 V DC nominal, maximum 6 V DC
Current Consumption	<100 mA continuous operation
Acquisition Time	< 0.02 sec
Rep Rate	1-100 Hz
Interface	I2C or PWM
Total size	40 x 48 x 20 mm
Weight	0.77oz. (22 g)

A.7 Air Speed Sensor [31]

Sensor Type	Measurement Specialties 4525DO
Pressure Range	1 psi (roughly up to 100 M/s or 360 Km/h or 223 Mp/h)
Resolution	0.84 Pa
Data Rate	14 bits from a 24-bit delta sigma ADC.

A.8 PPM Encoder [32]

Model	ATmega 328 µP
Size	22 x 19 x 5.5 mm
Weight	0.05 oz. (1.45 g)

A.9 Servo Multiplexer [33]

Size	0.97 × 0.97 in.
Weight	0.15 oz. (4.3 g)
Minimum operating voltage	2.5 V
Maximum operating voltage	16 V

A.10 Motor Speed Controller [34]

Input voltage	3S–4S Li-Po, 9- to 12-cell Ni-MH/Ni-Cd input voltage
Input connector types	16 AWG with E-flite® EC3™ connector
Output connector types	16 AWG with 3.5 mm female gold bullet connectors
BEC voltage	5.5 V switch mode, 700 mA continuous
Brake	Yes
Continuous maximum current	30 A
Momentary peak current	35 A
Length	2.0 in (51 mm)
Width	1.1 in (28 mm)
Height	0.35 in (8.7 mm)
Weight	1.1 oz (31 g)

A.11 Motor [35]

Voltage	3S LiPoly
RPM/V	840 kV
Weight	5.4 oz. (152 g)

A.12 Battery [36]

Type	Lithium polymer
Capacity	3200 mAh
Voltage	11.1 V
Wire gauge	13
Number of cells	3
Weight	8.45 oz .(239.5 g)
Configuration	3 S
Length	5.20 in (131 mm)
Width	1.71 in (43 mm)
Height	0.75 in (19 mm)
Maximum continuous discharge	20 C
Maximum continuous current	64 A

A.13 Servos [37], [38]

Model	Eflr7150	Eflr7155
Weight	1.6 oz. (45 g)	0.8 oz. (22 g)
Size	4 x 2 x 1.2 in.	3.9 x 2.1 x 0.6 in.
Torque	37 g	13 g
Gear Train	Plastic	Plastic
Plug	JR/Futaba	JR/Futaba

A.14 Radio Control Receiver [39]

Channels	8
Modulation	FM (Dual)
Frequency	72 MHz
Range	5500 feet
Voltage	4.8 - 6.0 V
Weight	0.81 oz. (23 g)
Size	0.30 x 1.4 x 0.8 in.

Appendix B

B.1 Bode Plot Examples

$$\mathrm{dB} = 20\log_{10}\mathrm{M_G}$$

$\mathrm{M_G}$ - Magnitude of $\mathrm{G_{(s)}}$

Magnitude

$$\mathrm{M_G} = |\mathrm{G}(\mathrm{j}\omega)|$$

Phase shift

$$\phi_{\mathrm{G}} = \angle \mathrm{G}(\mathrm{j}\omega)$$

When

$$\mathrm{G_{(s)}} = \mathrm{S}$$

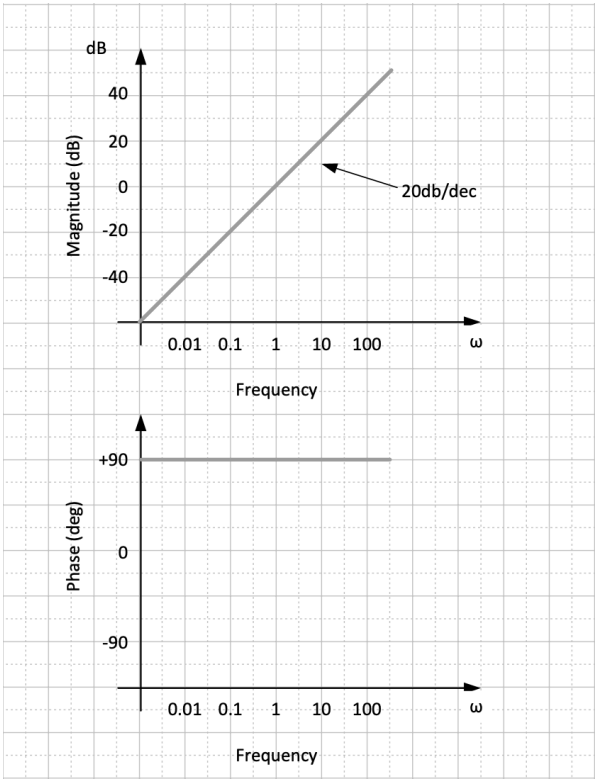


Fig. B1. Bode plot example s function.

When

$$G_{(s)} = \frac{1}{S}$$

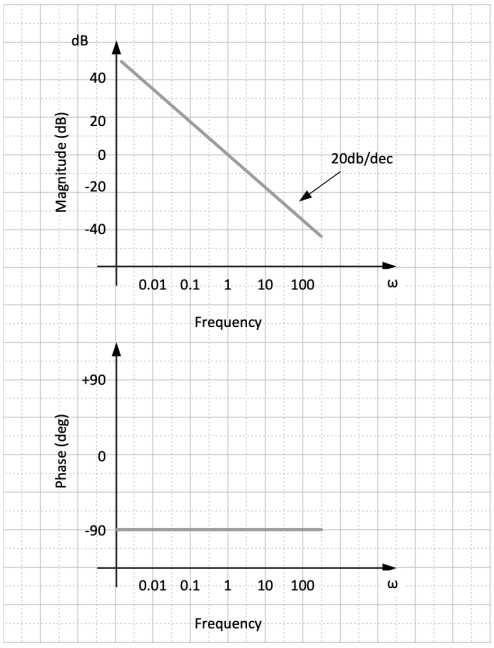


Fig. B2. Bode plot example 1/s function.

When

$$G_{(s)} = s + a$$

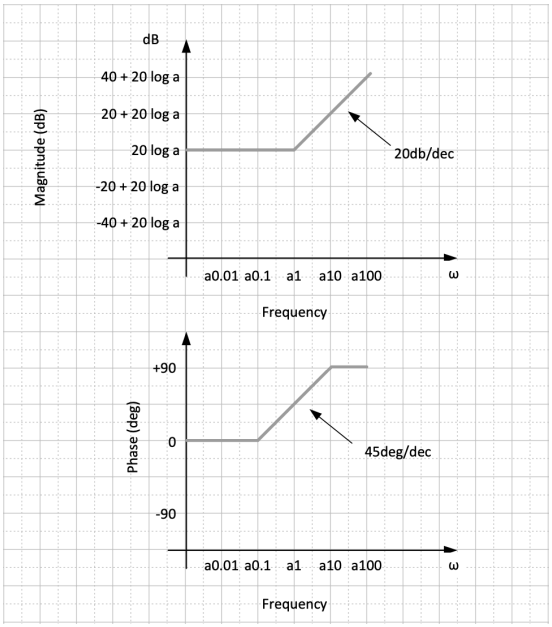


Fig. B3. Bode plot example (s+a) function.

When

$$G_{(s)} = \frac{1}{s + a}$$

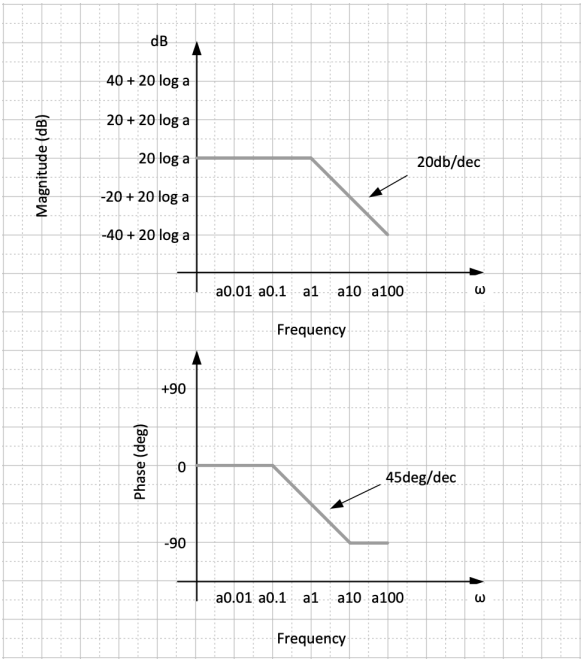


Fig. B4. Bode plot example 1/(s+a) function.

Appendix C

C.1 Gain and Phase Margin Example

For the example in Fig. 63, the gain and phase margin were calculated as follows:

$$dB = 20\log_{10} M_G$$

M_G - Magnitude of $G_{(s)}$

Magnitude

$$M_G = |G(j\omega)|$$

Phase shift

$$\phi_G = \angle G(j\omega)$$

When

$$G_{(s)} = S$$

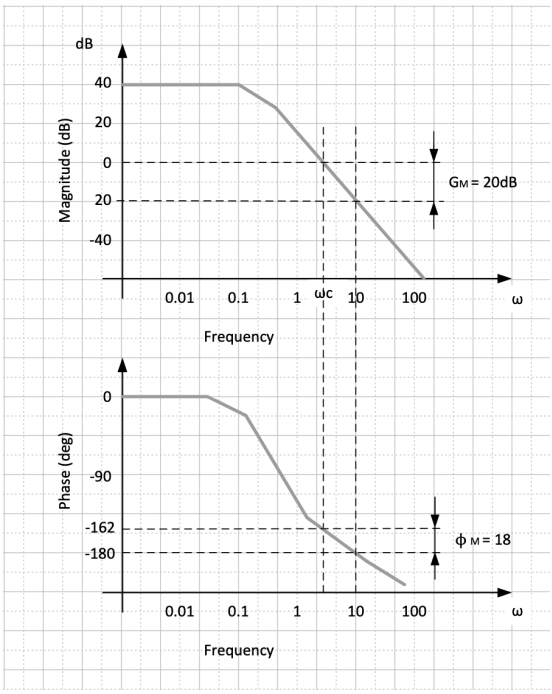


Fig. C1. Phase and gain margin example.

Phase margin: $\phi_M = \angle G(j\omega) + 180$ = -162 + 180 = 18°

Gain margin: G_M = 20 dB

Appendix D

D.1 Poles and Zeros Overview

Poles of a transfer function are the values of S that make the transfer function become infinite. Zeros of a transfer function are the values of S that make the transfer function become zero.

For the following transfer function:

$$G(s)=\frac{b_0 s^m+b_1 s^{m-1}+\cdots+b_{m-1}s+b_m}{a_0 s^n+a_1 s^{n-1}+\cdots+a_{n-1}s+a_n}$$

Zeros are the value of S that makes the numerator equal to zero, as such the transfer function value will be zero.

$$b_0 s^m + b_1 s^{m-1} + \cdots + b_{m-1}s + b_m = 0$$

Poles are the value of S that makes the denominator equal to zero, as such the transfer function value will be infinite.

$$a_0 s^n + a_1 s^{n-1} + \cdots + a_{n-1}s + a_n = 0$$

For example, for the transfer function:

$$G(s)=\frac{s+1}{s^2+7s}$$

Where

$$s + 1=0$$

The zero is

$$s=-1$$

Where

$$s^2 + 7s=0$$

The poles are

$$s=0,\ s=-7$$

The results can be displayed on a poles and zeros graph as shown in Fig. 81:

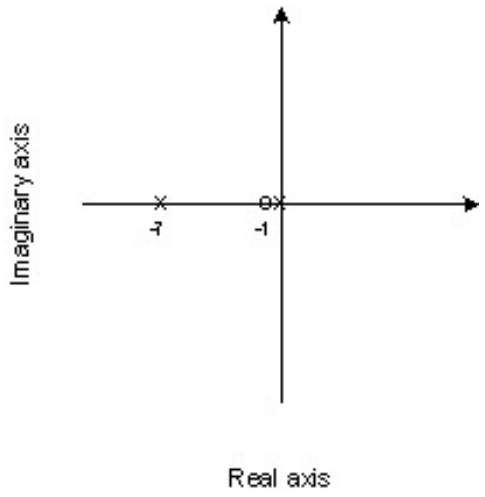


Fig. D1. The graph of the poles and zeros.

D.2 Poles and Zeros Simplification

Pole-zero simplification is a MATLAB tool that is capable of reducing the order of a transfer function. The tool can cancel pole-zero pairs or states that do not have any effect on the model response. Pole-zero pairs can result from the construction of a closed loop system. This simplification preserves model response characteristics. Several types of pole-zero simplifications are supported. Structural elimination utilizes elimination of states that are disconnected structurally from the inputs or outputs. This process is efficient as it does not require any numerical computation. It does not change the state structure of the connected states. Another type of simplification is pole-zero cancellation, or minimal realization. This simplification eliminates unobservable or uncontrollable states from the state space models.

By simplifying the order of poles and zeros, the mathematical problem becomes less complex. The simplified function will have fewer coefficients and

multipliers, resulting in a simpler software implementation requiring less computational power. The graph below shows a MATLAB generated Bode plot and pole-zero maps for a reduced transfer function.

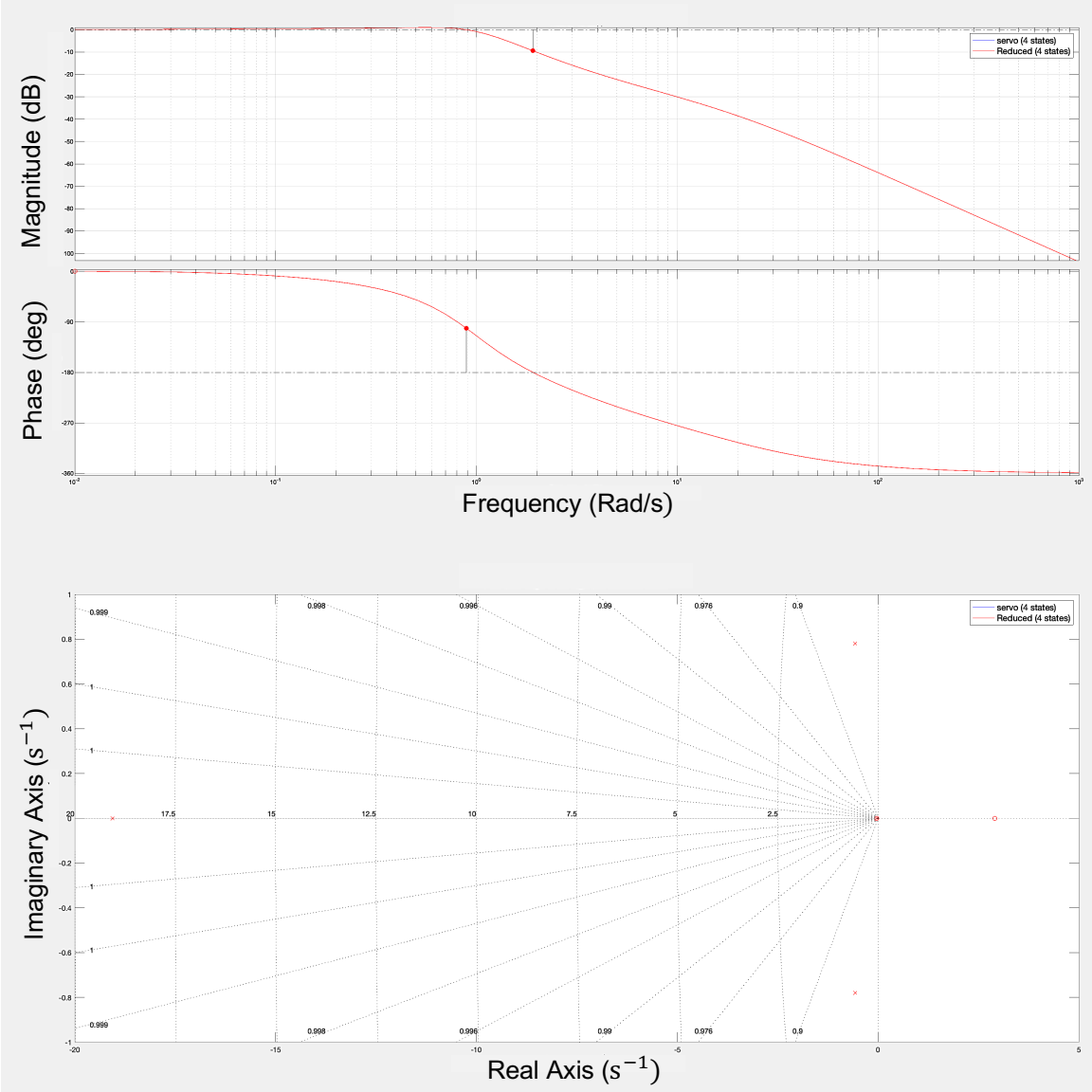


Fig. D2. Bode plot of reduced pole and zero.